第2章 FPGA活用回路&HDLサンプル記述13連発！

川岡圭一，河野崇，相田泰志，田中良平，松石剛啓，吉田政和

FPGAを活用した13種類のサンプル回路を紹介する．ここで紹介するのは，FPGAを使ったボード・レベルの回路図と，FPGAの内部を記述したHDLソース・コードである．所望の信号を入出力するためには，多くの場合，FPGAの外部にいくつかの部品を組み合わせて構成した回路が必要になる．それを制御するのは当然FPGAであり，その機能はHDLで記述することになる．実際の設計や学習の参考回路として，本サンプル回路をおおいに役立ててほしい．（編集部）

本稿で紹介するサンプル回路を下掲の**表**にまとめました．大きく分類すると，「ディジタル入力回路関連」が3件，「ディジタル出力回路関連」が3件，「アナログ入力回路関連」が1件，「アナログ出力回路関連」が3件，「インターフェース回路関連」が3件になっています．

すべてのサンプル回路について，FPGAとその外付け部品で構成する回路図，およびFPGAに実装する機能ブロックのHDL記述を紹介しています．ただし，回路図において，FPGAの動作に共通して必要となる電源回路やコンフィグレーション回路，クロックなどは省略しています．また，単独で動作する完全な回路図にはなっていません．図には，FPGAのI/Oを介して外部と接続するポートと，FPGAの内部でほかの機能ブロックとつなぐポートを示してあります．HDL記述についてはできるだけ誌面に示しています．ただし，ソース・コードが長くなってしまうものについては，本誌のホームページ（http://www.cqpub.co.jp/dwm/）からダウンロードできるようにしました．

ここで紹介する回路は，記事中で示した環境で実際に動作しているものです．ただし，評価のための簡易的な機能しか持たないものもあれば，実際の設計で活用した機能ブロックをそのまま紹介しているものもあります．回路の設計趣旨が記事ごとに異なることをあらかじめご了承ください．

表 本稿で紹介するサンプル回路

	回路名	分　類	主要部品	ページ
No.1	スイッチ入力回路	ディジタル入力	機械式スイッチ	P.29
No.2	4×4マトリックス・スイッチ入力回路	ディジタル入力	機械式スイッチ	P.32
No.3	外部機器からのディジタル入力回路	ディジタル入力	フォトカプラ	P.34
No.4	外部機器へのディジタル出力回路	ディジタル出力	リレー，フォトカプラ	P.36
No.5	7セグメントLEDのダイナミック点灯回路	ディジタル出力	7セグメントLED	P.38
No.6	2色ドット・マトリックスLED点灯回路	ディジタル出力	ドット・マトリックスLED	P.41
No.7	アナログ入力回路	アナログ入力	A-Dコンバータ	P.45
No.8	アナログ出力回路	アナログ出力	抵抗，コンデンサ	P.48
No.9	高速アナログ出力回路	アナログ出力	D-Aコンバータ	P.50
No.10	4チャネル・アナログ出力回路	アナログ出力	D-Aコンバータ	P.52
No.11	同期SRAMインターフェース回路	インターフェース	ZBT-SSRAM	P.54
No.12	PCカード・インターフェース回路	インターフェース	CPLD	P.56
No.13	I^2C-busインターフェース回路	インターフェース	抵抗，FET	P.60

01 スイッチ入力回路

松石剛啓

外部回路

- 回路図：図1-1
- 主要部品：機械式スイッチ(タクタイル・スイッチなど)

HDL記述

- VHDL記述：リスト1-1
- 外部入力：sw_pulse_in
- 外部出力：なし
- 内部入力：clk，rstn
- 内部出力：sw_pulse_out
- パラメータ：cha_cnt_max
  (チャタリング除去用カウンタの最大値)

評価環境

ISE WebPACK 7.1i，Spartan-IIE

多くの機器では，入力デバイスとしてスイッチが用いられています(**写真1-1**)．スイッチの状態や変化によって，LSIの動作を切り替えるという使いかたが一般的です．

FPGAにスイッチを接続するもっとも簡単な回路を**図1-1**に示します．スイッチがOFF(接点が開いている)のときには，FPGAには“H”が入力されています．スイッチをONにすると，FPGAには“L”が入力されます．

“H”か“L”が入力されるだけなので，HDL記述は簡単そうです．しかし，機械式のスイッチを使うときには注意しなければならないことがあります．

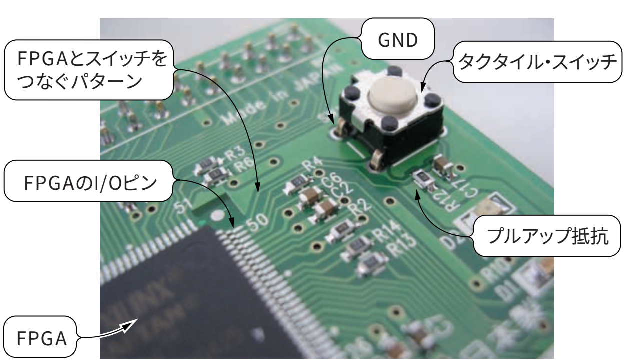


写真1-1　FPGAに接続されたスイッチ

プリント基板に実装されたFPGAとスイッチ．FPGAのI/Oピンにスイッチが接続されていることがわかる．

● 機械式スイッチではチャタリングが発生

機械式のスイッチやリレーは，接点が切り替わるとき，接点部のバウンドにより短い時間にON/OFFを繰り返すチャタリング(バウンシングともいう)という現象が発生します(**図1-2**)．このチャタリングを考慮せずに回路を設計すると，誤動作してしまうことがあります．

例えば，スイッチをONした回数を数えたいとき，チャタリングが発生すると，1回のONでも数回と認識してしまいます．

チャタリングの時間は，スイッチやリレーの種類によって異なります．一般的なスイッチでは数msのものが多いようです．これはデータシートに明示されているので，使用する部品の特性を確認して設計しなければなりません．

● カウンタを使ってチャタリングを除去

チャタリングの影響を除去するためには，いくつかの方法があります．ここでは，一定時間ON/OFFが続いたときのみON/OFFと判断する方法を考えます．

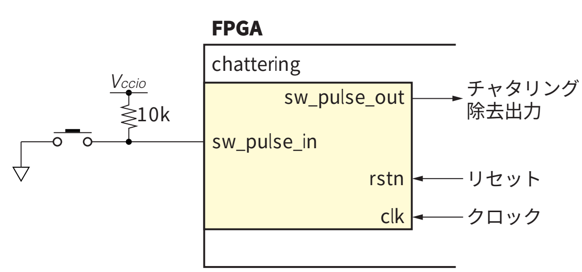


図1-1　FPGAにスイッチを接続するもっとも簡単な回路

スイッチがOFFのときには，FPGAには“H”が入力されている．スイッチをONにすると，FPGAには“L”が入力される．

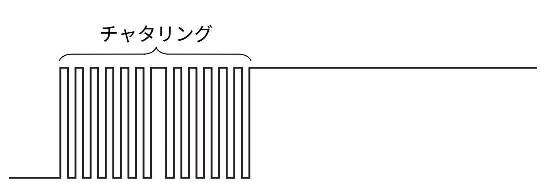


図1-2　チャタリングの波形

機械式のスイッチやリレーは，接点が切り替わるとき，接点部のバウンドによって短い時間にON/OFFを繰り返す．

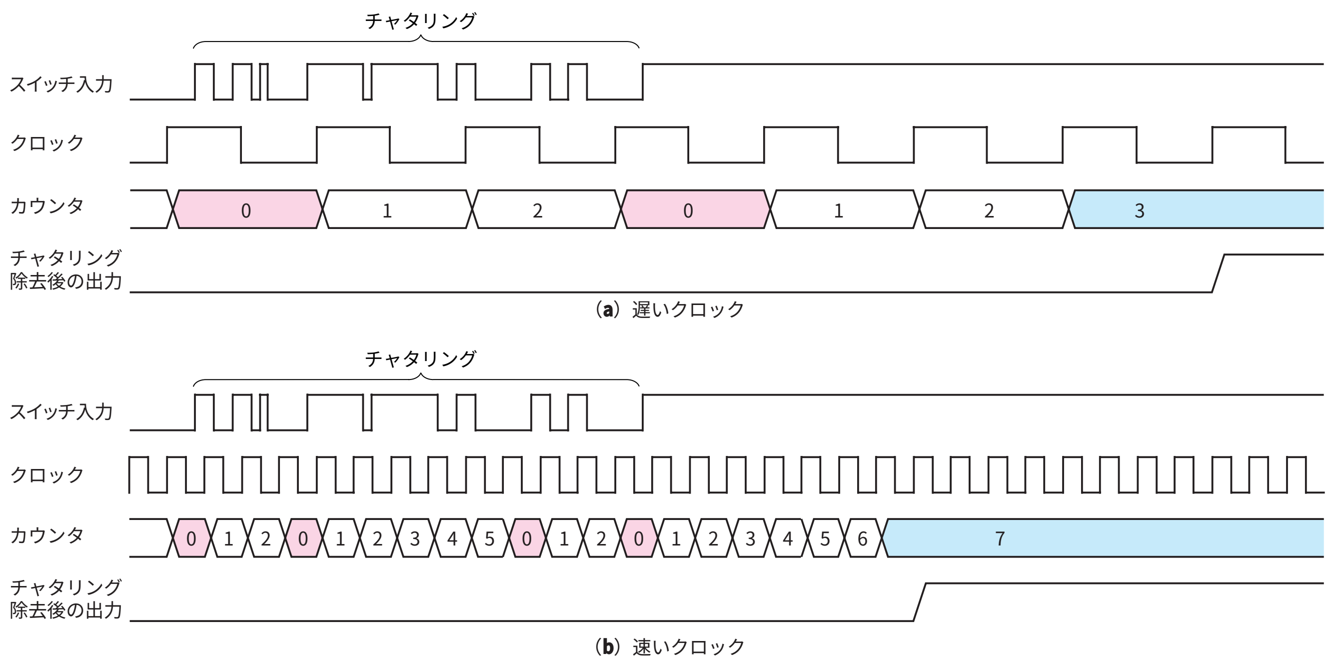


図1-3　チャタリング除去タイミング

スイッチからの入力信号が"H"のとき，カウンタをインクリメントし，"L"になったらカウンタをクリアする．カウンタが最大値になったら，カウントを停止する．カウンタが最大値に達しているときに"H"，それ以外のときに"L"を出力すれば，チャタリングを除去できる．カウンタの最大値は，クロック周波数によって調整する．

リスト1-1　チャタリング除去機能を持たせたスイッチ入力ブロックのVHDL記述

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity chattering is
  Port (
    clk          : in  std_logic;   -- クロック
    rstn         : in  std_logic;   -- リセット
    sw_pulse_in  : in  std_logic;   -- 外部スイッチからの入力
    sw_pulse_out : out std_logic);  -- チャタリング除去後の出力
end chattering;

architecture rtl of chattering is

constant cha_cnt_max : integer := 1023;   -- カウンタの最大値
signal chatter_hi_cnt : integer range 0 to cha_cnt_max;   -- "H"を数えるカウンタ
signal chatter_low_cnt : integer range 0 to cha_cnt_max;  -- "L"を数えるカウンタ
signal sw_pulse_out_tmp : std_logic;

begin

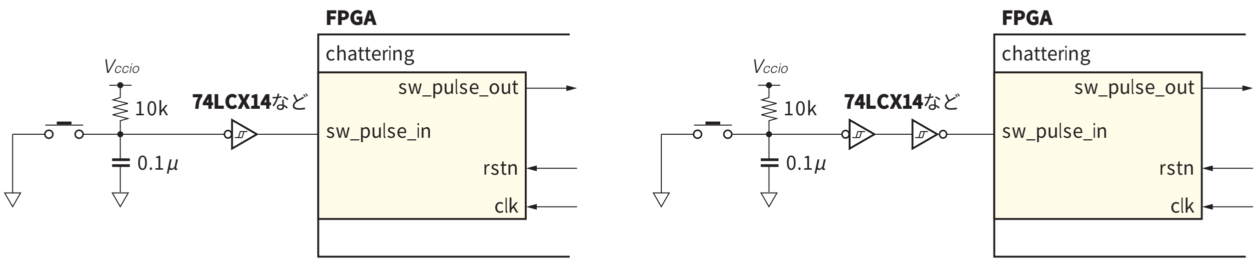
u_chatter_hi_cnt : process (rstn, clk)   -- "H"を数える
begin
  if (rstn='0') then
    chatter_hi_cnt <= 0;
  elsif (clk'event and clk='1') then
    if (sw_pulse_in='1') then
      if (chatter_hi_cnt=cha_cnt_max) then          -- カウンタが
        chatter_hi_cnt <= chatter_hi_cnt;            -- 最大値に
      else                                            -- なったら
        chatter_hi_cnt <= chatter_hi_cnt + 1;        -- カウントを停止
      end if;
    else
      chatter_hi_cnt <= 0;
    end if;
  end if;
end process;

u_chatter_low_cnt : process (rstn, clk)   -- "L"を数える
begin
  if (rstn='0') then
    chatter_low_cnt <= 0;
  elsif (clk'event and clk='1') then
    if (sw_pulse_in='0') then
      if (chatter_low_cnt=cha_cnt_max) then         -- カウンタが
        chatter_low_cnt <= chatter_low_cnt;          -- 最大値に
      else                                            -- なったら
        chatter_low_cnt <= chatter_low_cnt + 1;      -- カウントを
      end if;                                         -- 停止
    else
      chatter_low_cnt <= 0;
    end if;
  end if;
end process;

u_sw_pulse_out_tmp : process (rstn, clk)
begin
  if (rstn='0') then
    sw_pulse_out_tmp <= '0';
  elsif (clk'event and clk='1') then
    if (chatter_hi_cnt=cha_cnt_max) then       -- "H"を数えるカウンタが最大値
      sw_pulse_out_tmp <= '1';                 -- のとき"H"を出力
    elsif (chatter_low_cnt=cha_cnt_max) then   -- "L"を数えるカウンタが
      sw_pulse_out_tmp <= '0';                 -- 最大値のとき"L"を出力
    else
      sw_pulse_out_tmp <= sw_pulse_out_tmp;    -- 前の状態を保持
    end if;
  end if;
end process;

u_sw_pulse_out : sw_pulse_out <= sw_pulse_out_tmp;

end rtl;
```



(a) スイッチONのとき“H”　(b) スイッチONのとき“L”

図1-4　外部回路によってチャタリングの発生を抑える
積分回路とスレッショルド特性を持つバッファによって微小時間の信号変化を除去する．

例えば，使用するスイッチのチャタリング時間を3msとします．入力信号が“L”から“H”に変化してから3ms以上“H”が連続したら，入力信号が“H”になったと確定します．3ms以内に“L”に戻った場合は，入力信号は“H”にならなかったと判断します．

この動作を実現するには，カウンタ回路を用います．システム・クロックをカウントして必要な時間を数えればよいのです．ここでは，アップ・カウンタを用いることにします．

図1-3のように，スイッチからの入力信号が“H”のとき，カウンタをインクリメントし，“L”になったらカウンタをクリアします．すると，入力信号が安定していないとき（チャタリング発生時）は，カウンタはインクリメントとクリアを繰り返し，入力信号が安定するとインクリメントのみとなります．カウンタが最大値になったら，カウントを停止します．同様に，スイッチからの入力信号が“L”のときにインクリメントするカウンタを設けます．スイッチからの入力信号が“H”のときインクリメントするカウンタが最大値に達しているときに“H”，スイッチからの入力信号が“L”のときインクリメントするカウンタが最大値に達しているときに“L”，それ以外のときは前の状態を保持するように出力すれば，チャタリングを除去できます．

カウンタがオーバフローしてしまうと，安定した信号が入力されているにもかかわらず，信号が変化してしまうので注意してください．カウンタの最大値は，ON/OFFを判定する時間で，チャタリング時間以上になるように設定すると確実です．

カウンタ回路を動作させるシステム・クロックの周波数によって，カウンタの最大値の設定が変わります．例えば，50MHzのクロックで3msを数えようとすると，最大値を150,000にします．しかし，2kHzのクロックだと最大値は6になります．したがって，システム・クロックが遅いほうが，小規模の回路ですみます．しかし，機器の性能に影響を及ぼすことになるかもしれません．

リスト1-1にチャタリング除去機能を持たせたスイッチ入力ブロックのVHDL記述を示します．カウンタの最大値cha_cnt_maxは，システム・クロックの周波数と使用するスイッチのチャタリング時間に合わせて設定します．

● 外部回路によるチャタリング対策

ここでは，FPGA内部の回路によってチャタリングを除去する方法について説明しました．

このほか，外部回路によってチャタリングの発生を抑えることもできます．簡単な例を図1-4に示します．積分回路とスレッショルド特性を持つバッファによって微小時間の信号変化を除去するしくみです．

まついし・たかひろ
(株)ダイヘン

02 4×4マトリックス・スイッチ入力回路

相田泰志

外部回路

- 回路図：図2-1
- 主要部品：機械式スイッチ（タクタイル・スイッチなど）

HDL記述

- VHDL記述：リスト2-1
- 外部入力：KEY_IN
- 外部出力：KEY_SEL
- 内部入力：CLK
- 内部出力：KEY_BUF
- パラメータ：なし

評価環境

ISE WebPACK 7.1i，Quartus II Web Edition 5.0，自作評価ボード

テン・キーのように多数のキー入力が必要な場合，スイッチごとに1ピンを使用してしまうと，多くのI/Oピンが必要になってしまいます．このようなとき，マトリックス方式のスイッチ入力を用いることで，少ない配線で多数のキーを読み込むことができます．

図2-1に4×4のマトリックス・スイッチ入力回路を示します．4×4のマトリックスで16個のスイッチを接続しています．スイッチの部分には，ダイオードを配置することで，キーを2個以上同時に押したときでも正しく認識できます．

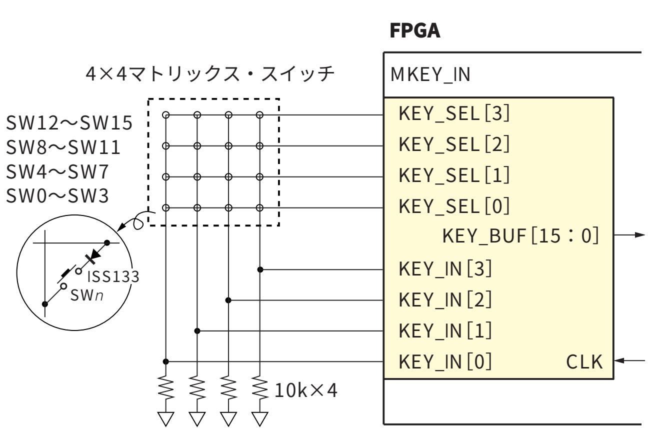


図2-1　4×4のマトリックス・スイッチ入力回路
4×4のマトリックスで16個のスイッチを接続する．

● 1行ずつ4回で読み込む

マトリックス状に並んだ16個のスイッチの状態を同時に読み込むことはできません．1行ずつ4回で読み込みます．

まず最初に，出力信号KEY_SEL[0]だけを“H”にします．KEY_SEL[3：1]は“L”です．この状態でスイッチ入力KEY_INを読み込むと，SW3～SW0の状態を知ることができます．これをバッファKEY_BUF[3：0]に保存しておきます．

次に，KEY_SEL[1]だけを“H”にし，読み込んだキーの状態をバッファKEY_BUF[7：4]に保存します．これはSW7～SW4の状態です．同じように，KEY_SEL[2]だけ“H”にしたときをKEY_BUF[11：8]に，KEY_SEL[3]だけ“H”にしたときをKEY_BUF[15：12]に保存します．

この動作を繰り返すことで，16個のキーの状態をつねに把握することができます．このVHDL記述を**リスト2-1**に示します．

● チャタリングへの配慮など

機械式スイッチでは，かならずチャタリングが発生します．そこで4クロック（すべてのキーの状態を読み込むために必要なクロック数）がチャタリング時間（一般には数ms）以上の周期になるようにします．FPGAの回路でチャタリングを除去しようとすると，とても複雑になります．

システム・クロックが高速すぎる場合には，分周したクロックでキーをスキャンしていくような修正が必要です．

なお，すべてのスイッチの状態を正しく得ることができるのはFPGAが起動してから4クロック以降なので，この点には注意が必要です．

そうだ・やすし
西山工業（株）

リスト2-1　4×4のマトリックス・スイッチ入力ブロックのVHDL記述

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all ;
use ieee.std_logic_arith.all ;

entity MKEY_IN is
  port (
      KEY_SEL : out std_logic_vector(3 downto 0);
                                      -- Key select
      KEY_IN :  in  std_logic_vector(3 downto 0);
                                       -- Key input
      KEY_BUF : out std_logic_vector(15 downto 0);
                                  -- key input buffer
      CLK :     in  std_logic     -- Sampling Clock
     );
end MKEY_IN;

architecture arch_MKEY_IN of MKEY_IN is

signal KEY_CNT: std_logic_vector(1 downto 0);

begin

process(CLK)
begin
    if (CLK'event and CLK = '0') then
        KEY_CNT <= KEY_CNT + 1;
        case KEY_CNT(1 downto 0) is
            when "00" =>   KEY_SEL <= "0001" ;
            when "01" =>   KEY_SEL <= "0010" ;
            when "10" =>   KEY_SEL <= "0100" ;
            when others => KEY_SEL <= "1000" ;
        end case ;
    end if;
    if (CLK'event and CLK = '1') then
        case KEY_CNT(1 downto 0) is
            when "00" =>   KEY_BUF(3 downto 0)  <= KEY_IN ;
            when "01" =>   KEY_BUF(7 downto 4)  <= KEY_IN ;
            when "10" =>   KEY_BUF(11 downto 8) <= KEY_IN ;
            when others => KEY_BUF(15 downto 12) <= KEY_IN ;
        end case ;
    end if ;
end process;

end arch_MKEY_IN;
```

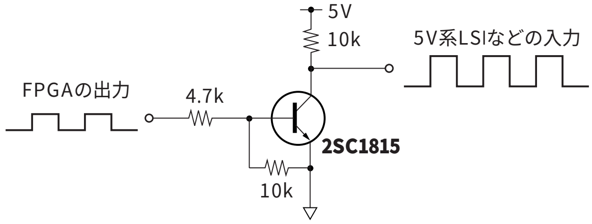
Column　5Vで動作するLSIとFPGAとのインターフェース

最近よく使われるFPGAは，5V系のI/O規格に対応していません．しかし，5Vで動作するLSIも少なからず存在しており，組み合わせて使いたいことがあります．ところがこれを直接接続すると，正常に動作しないばかりか，FPGAを破壊してしまうことさえあります．そこで使用するのがレベル変換回路です．

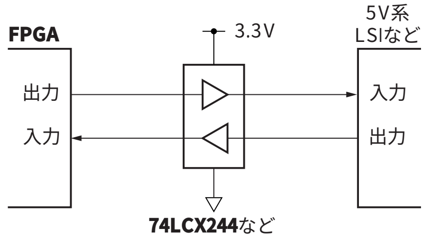
図A-1に，トランジスタ(またはFET)を使ったレベル変換回路の例を示します．FPGAの出力信号で利用できます．トランジスタのベースに電圧が加わると，トランジスタがONして，出力が0Vとなります．インバータとして動作するので，注意が必要です．

図A-2に標準論理ICを使ったレベル変換回路の例を示します．標準論理ICの多くは，入力側の電圧が電源電圧より高くても破壊されないような構造になっています．標準論理ICの電源電圧を，FPGAのI/O電圧に合わせれば，FPGAが直接受けられない信号を入力できるようになります．CMOSの標準論理ICは，"H"出力時の電圧は電源電圧近くになります．したがって標準論理ICの"H"出力時の電圧が5V系LSIの"H"入力の最低電圧(V_{IH})より高ければ，FPGAの出力信号にも使えます．

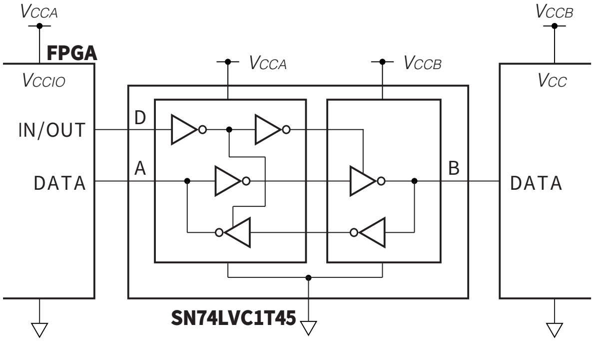
図A-3にレベル変換専用ICを使う例を示します．標準論理ICによるレベル変換では，5Vと1.8Vのような大きな電圧間のインターフェースを行うことはできません．レベル変換専用ICは，変換する2種類の電源を供給するので，電圧に差があってもレベル変換を行うことができます．ただし，2種類の電源の立ち上がりタイミングなどに注意する必要があります．5VのI/Oに対応し，複数のI/Oバンクを持つ一部のFPGAやCPLDも，この用途で使うことができます．



図A-1　トランジスタによる3.3V(1.8V，2V)→5V変換回路



図A-2　標準論理ICによる3.3V↔5Vレベル変換回路



図A-3　専用ICによるレベル変換回路

03 外部機器からのディジタル入力回路

相田泰志

外部回路

- 回路図：図3-1，図3-2，図3-4
- 主要部品：フォトカプラ

HDL記述

- VHDL記述：リスト3-1
- 外部入力：DIN
- 外部出力：なし
- 内部入力：CLK
- 内部出力：DATA
- パラメータ：なし

評価環境

ISE WebPACK 7.1i，Quartus II Web Edition 5.0，自作評価ボード

リスト3-1　ディジタル入力ブロックのVHDL記述

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all ;

entity D_IN is
  port (
    DATA : out std_logic;          -- Input Register
    DIN  : in  std_logic;          -- Digital input
    CLK  : in  std_logic;          -- Clock
     );
end D_IN;

architecture rtl of D_IN is

begin

process(CLK)
begin
    if (CLK'event and CLK = '1') then
        DATA <= DIN ;
      end if ;

end process;

end rtl;
```

ディジタル信号の入力は，“H”，“L”だけなので簡単に扱うことができます．しかし，外部機器から入力される信号の場合には，信号の電圧レベルや，FPGAの内部回路とタイミングを合わせるための処理が必要です．

ここではいくつかの外部回路を紹介します．FPGAの処理は，例えば**リスト3-1**のように入力信号をレジスタで受け，内部ブロックと同期させるだけです．

● 外部機器から信号を受けるときには保護回路が必要

外部からの信号を受ける場合，FPGAのI/O電圧より高い電圧や負の電圧などが印加されるおそれがあります．そのため，かならず保護回路を入れておく必要があります．

図3-1の回路では，I/O電源以上または0V以下の電圧が入力された場合，ダイオードによって超過分の電圧をカットします．高周波ノイズなどを避けるためにローパス・フィルタを挿入します．ヒステリシス特性を持つシュミット・トリガ回路を介することで，例えば入力信号の立ち上がりが非常にゆっくりだった場合に，しきい値付近で信号がばたつくのを防ぎます．

● 無電圧接点にはプルアップ抵抗を付ける

リレーやオープン・コレクタ出力のように，無電圧接点の入力を受ける回路例を**図3-2**に示します．無電圧接点の場合には，信号線をプルアップします．

また，オープン・コレクタ入力の場合は，ダイオードを入れる必要があります．これにより，電源をOFFにしたときに外部の機器から電流が流れ込み，異常動作になるというようなトラブルを防ぐことができます．

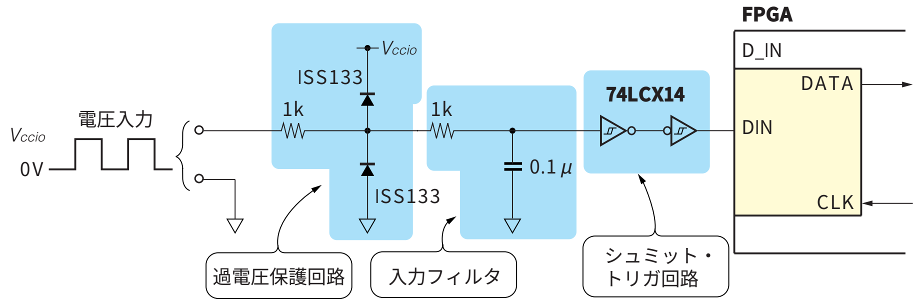
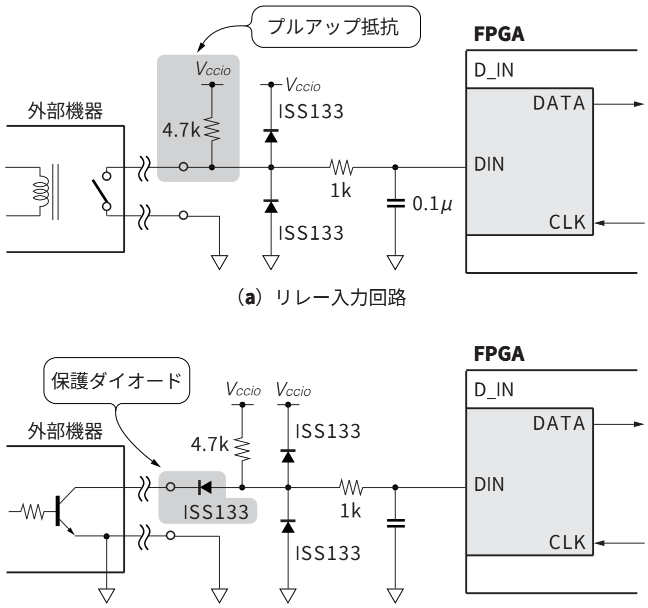


図3-1
電圧入力回路
I/O電源以上または0V以下の電圧が入力されてもLSIが破壊されないように，過電圧保護回路を設ける．



（a）リレー入力回路

（b）オープン・コレクタ入力回路

図3-2　無電圧入力回路
無電圧接点の入力では，信号線をプルアップする．

● フォトカプラとリレーによる絶縁回路

絶縁には，商用電源AC100Vなどから切り離して感電しないようにする基礎絶縁と，機器間で行う信号絶縁の二つがあります．

図3-3のように二つの機器が動作しているとき，ノイズなどがあればそのまま信号に重畳される形になります．ノイズの電圧が大きければ，正常に信号を認識できなかったり，電流の流れ込みによるLSIの破壊なども起こしかねません．そこで絶縁によって信号成分だけを伝送します．

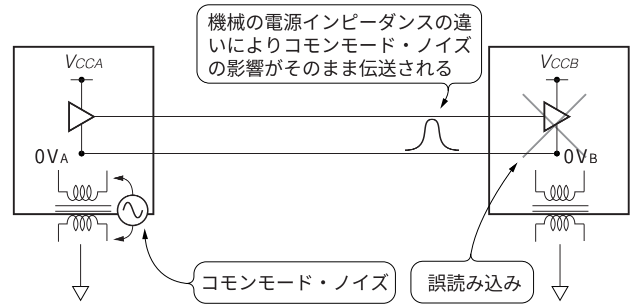
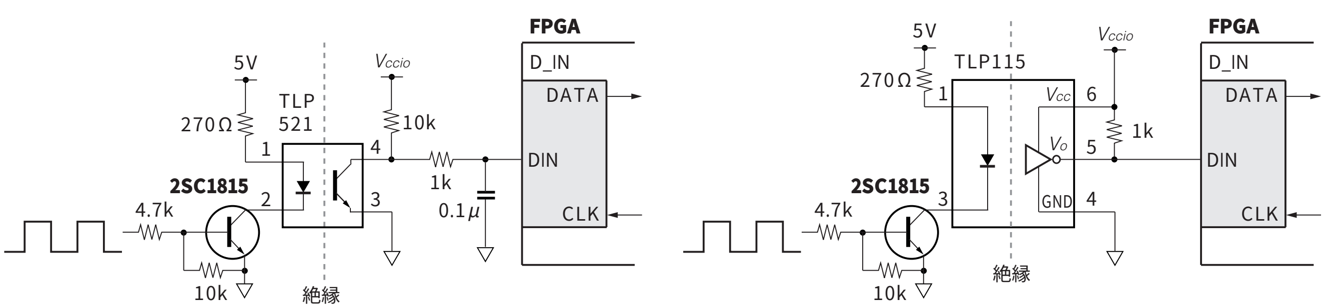


図3-3　ノイズの影響
ノイズの電圧が大きければ，正常に信号を認識できなかったり，電流の流れ込みによるLSIの破壊などが起こる可能性もある．

図3-4にフォトカプラを使用した絶縁回路を示します．フォトカプラによって回路が電気的に分離されることになります．フォトカプラには，トランジスタ出力のものや高速信号向けのIC内蔵型まで多くの種類があるので，用途に合わせて選択することが可能です．

絶縁用の部品としてリレーを使うこともあります．ただし，機械的な部品のためチャタリングの影響があり，動作速度が遅くなります．

そうだ・やすし
西山工業（株）



（a）トランジスタ出力のフォトカプラ

（b）IC内蔵型のフォトカプラ

図3-4　フォトカプラを使用した絶縁回路
フォトカプラによって回路が電気的に分離され，信号成分のみを伝えることができる．

04 外部機器へのディジタル出力回路

相田泰志

外部回路

- 回路図：図4-1，図4-2，図4-4
- 主要部品：リレー，フォトカプラ

HDL記述

- VHDL記述：リスト4-1
- 外部入力：なし
- 外部出力：DOUT
- 内部入力：CLK，DATA
- 内部出力：なし
- パラメータ：なし

評価環境

ISE WebPACK 7.1i，Quartus II Web Edition 5.0，自作評価ボード

ディジタル出力回路というと，まず，外部の機器をON/OFFする用途が思いうかびます．この回路にはロボットのモータや機器の電源，ランプのON/OFFなど，さまざまな用途があります．

I/OポートのON/OFFに従って負荷を駆動する回路を制御すれば，外部の機器を動作させることができます．FPGAによる制御は，I/OをON/OFFするだけなので簡単です（**リスト4-1**）．

● モータやランプのON/OFFにはリレーを使う

モータやランプのように，高い電圧，大きな電流を必要とする部品を電子回路で制御する際には，リレーを使うのが簡単です．**図4-1**にリレーのON/OFF回路の例を示します．

まず，トランジスタによりリレーをONします．これはリレーを動作させるためには，大きな電流が必要になった

リスト4-1　ディジタル出力ブロックのVHDL記述

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity D_OUT is
  port (
    DATA : in  std_logic;
               -- Input Register
    DOUT : out  std_logic;
               -- Digital input
    CLK  : in  std_logic
               -- Clock
     );
end D_OUT;

architecture rtl of D_OUT is

begin

process(CLK)
begin
    if (CLK'event and CLK = '1') then
       DOUT <= DATA ;
      end if ;

end process;

end rtl;
```

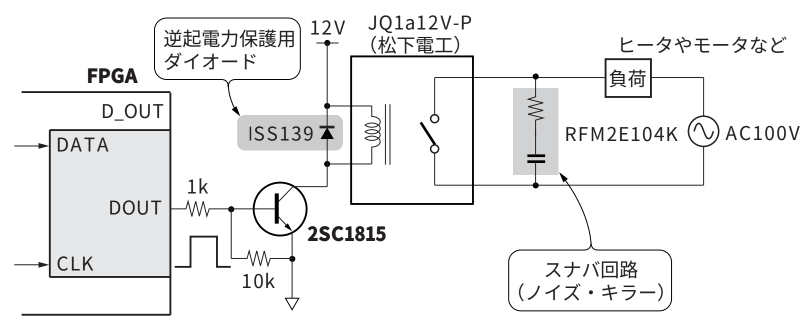


図4-1　リレー出力回路

I/Oポートを"H"にするとリレーのスイッチがON，"L"にするとリレーのスイッチがOFFになる．この回路は，モータやランプのように，高い電圧，大きな電流を必要とする部品を電子回路で制御する際に使用できる．

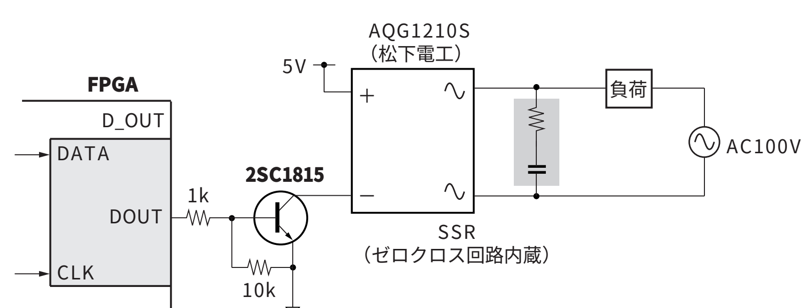


図4-2　ソリッドステート・リレー出力回路

ソリッドステート・リレーは，フォトトライアック・カプラやトリガ回路などで構成される半導体リレーである．AC電圧の負荷をON/OFFするときに使用する．

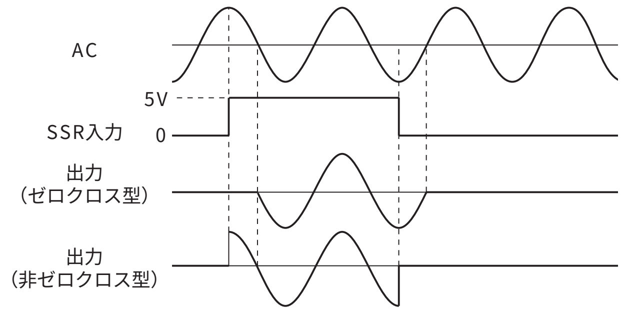


図4-3　ソリッドステート・リレーの動作
非ゼロクロス型は，入力側のON/OFFに合わせて出力をON/OFFする．ゼロクロス型は，入力側がON/OFFした後で，出力側のAC電圧が0VになったときにON/OFFする．

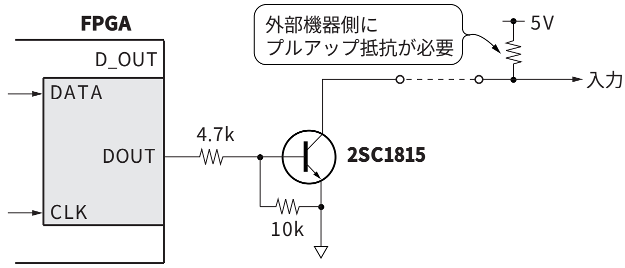


図4-4　オープン・コレクタ出力回路
FPGAのI/O電圧より高い電圧を扱いたい場合のレベル変換やワイヤードORを使いたいときに便利．

り，FPGAのI/O電圧より高い12Vのような電圧を必要とするためです．最近では5V程度の電圧や小さな電流で動作するリレーもありますが，リレーを接続する場合はトランジスタでスイッチングしたほうが無難です．リレーの駆動電圧は高いほうが駆動電流を抑えることができるので，省電力化の面では有利になります．

そしてリレーをON/OFFするときに発生する逆起電力を逃がすため，リレーのコイルの両端にダイオードを接続しておきます．

また，リレーの出力側はON/OFFによるノイズを除去するために，スナバと呼ばれる*RC*の複合素子を接点と並列に挿入しておきます．

リレーでON/OFFできる回路は，リレーの特性（接点容量）によって決まります．抵抗負荷（ヒータなど）では1.5～2倍，誘導負荷（モータ，電球など）では5倍くらいの余裕が必要です．定格と比べてリレーに流す電流が少なすぎると，接触不良が起こることもあるので，負荷に合わせた適切な容量を選ぶことが重要です．

なお，リレーは機械的な動作を行うため，部品の寿命に注意が必要です．短い周期でON/OFFさせて使用する場合などは，比較的短い期間で寿命を迎えてしまうかもしれません．

● AC電圧のON/OFFではSSRを使う

AC電源のON/OFFを行う場合には，ソリッドステート・リレー（SSR：solid-state reley）が便利です．**図4-2**にソリッドステート・リレーによる回路例を示します．

ソリッドステート・リレーには，ゼロクロス型と非ゼロクロス型があります．非ゼロクロス型は，入力側のON/OFFに合わせて出力もON/OFFします．ゼロクロス型は，入力側がON/OFFした後で，出力側のAC電圧が0VになったときにON/OFFするので，ON/OFF時のノイズを抑えることができます（**図4-3**）．

リレーをON/OFFさせる場合には，ON/OFFに時間がかかります．とくに，ゼロクロス型のソリッドステート・リレーでは，ゼロクロスがこないと切り替わらないため，応答速度などに注意する必要があります．

● 非絶縁ならオープン・コレクタ出力

内部の機器や非絶縁の信号を切り替えるときに便利なのが，オープン・コレクタ方式の出力です．**図4-4**に回路を示します．

リレーやソリッドステート・リレーと比べて速い切り替えができます．FPGAのI/O電圧より高い電圧を扱いたい場合のレベル変換も兼ねることができます．ワイヤードORで使うことができるので，複数の信号で制御する出力などにも便利です．

また，絶縁する場合には，フォトカプラを利用した出力回路が利用できます．

そうだ・やすし
西山工業（株）

05 7セグメントLEDのダイナミック点灯回路

松石剛啓

外部回路	
●回路図：図5-1	
●主要部品：7セグメントLED	
HDL記述	
●VHDL記述：リスト5-1	
●外部入力：なし	
●外部出力：led_sel，led_data	
●内部入力：led1_data，led2_data，led3_data，led4_data，clk，rstn	
●内部出力：なし	
●パラメータ：なし	
評価環境	
ISE WebPACK 7.1i，Spartan-II，Spartan-II E300	

よく使われる表示デバイスの一つに，7セグメントLEDがあります．7個のLEDがディジタルの「8」の字型に配置されているモジュール部品です．多くの7セグメントLEDは，小数点表示に使える「.（ドットポイント）」の表示部分を持ちます．

● 残像効果を利用するダイナミック点灯方式

7セグメントLEDは使いやすい部品ですが，けた数を増やそうとすると，どうしても必要なI/Oピンが増えてしまいます．1けたで8本のI/Oピンを使用するので，8×けた数のI/Oピンが必要になります．また，回路設計にもよりますが，LEDを明るく点灯させようとすると1セグメント当たり10mA～20mAの電流を必要とします．1けた分のセグメントがすべて点灯すると80mA～160mAということになり，大きな電流が必要になります．

そこで，複数けたの7セグメントLEDでは，ダイナミック点灯方式がよく用いられます．これはすべての7セグメントLEDを一斉に点灯させるのではなく，1けたずつ高速に切り替えながら点灯するものです．残像効果により，人間の目にはすべての7セグメントLEDがついているように見えます．また，つねに1けた分のLEDしか点灯しないので，消費電力も抑えられます．

● 表示パターンと表示けたを出力

4けたの7セグメントLEDをダイナミック点灯する回路の例を**図5-1**に示します．7セグメントLEDとしては，アノード・コモンの「TLG337T（東芝）」を使用しています．また，FPGAとLEDは，ドライバICの「TD62084（東芝）」を介して接続しています．

FPGAからは，7セグメントLEDの表示パターンを示すled_data［7：0］と，点灯させる7セグメントLEDを選択するled_sel［3：0］を出力します．

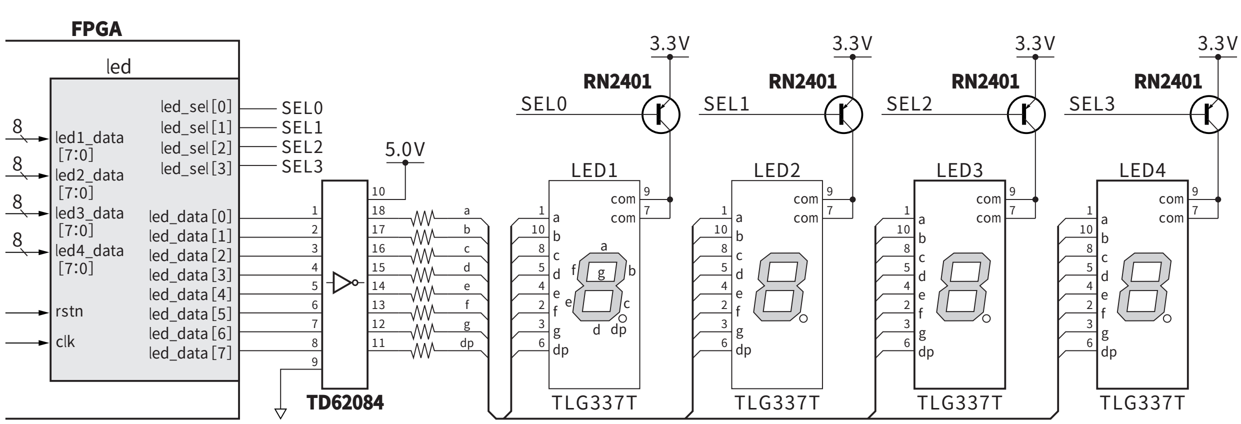


図5-1　7セグメントLEDダイナミック点灯回路

7セグメントLEDの表示パターンを示すled_data［7：0］と，点灯させる7セグメントLEDを選択するled_sel［3：0］で制御する．ドライバICがインバータの働きをするので，led_data［7：0］は“H”で点灯，led_sel［3：0］は“L”で点灯である．

使用している7セグメントLEDでは，“L”を入力したセグメントが点灯します．TD62084は“H”を入力したときに“L”を出力するインバータの働きをします．したがって，led_data[7：0]から“H”を出力したビットが点灯することになります．

led_sel[3：0]は，7セグメントLEDのコモン(com)ピンを制御します．comピンには大きな電流が流れるため，トランジスタでスイッチングします．led_sel[n]が“L”のときに対応した7セグメントLEDのcomピンがONになり，LEDが点灯します．

4けたの7セグメントLEDをダイナミック点灯するブロックのVHDL記述を**リスト5-1**に示します．

led_sel[3：0]は，**図5-2**のように，1本ずつ順番に“L”を出力します．これは，“0111”の値を1ビットずつ左へローテートすることで実現でき，LED1，LED2，LED3，LED4と順番に点灯させられます．led_sel[3：0]をすべて“L”にすると，7セグメントLEDは四つとも同じ表示になります．ただし，通常の4倍の電流が流れることになり，電源容量の不足による誤動作につながることがあるので注意してください．

7セグメントLEDへのデータ出力は，led_sel[0]が“L”のときにLED1のデータを出力，led_sel[1]が“L”のときにLED2のデータを出力となるように記述します．

リスト5-1　7セグメントLEDダイナミック点灯ブロックのVHDL記述

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity led is
  Port (
    clk       : in  std_logic;    -- Clock
    rstn      : in  std_logic;    -- /Reset(Active Low)
    led1_data : in  std_logic_vector(7 downto 0); -- LED1 Data
    led2_data : in  std_logic_vector(7 downto 0); -- LED2 Data
    led3_data : in  std_logic_vector(7 downto 0); -- LED3 Data
    led4_data : in  std_logic_vector(7 downto 0); -- LED4 Data
    led_sel   : out std_logic_vector(3 downto 0); -- LED_SEL
    led_data  : out std_logic_vector(7 downto 0));-- 7segment Data
end led;

architecture rtl of led is

signal led_sel_s : std_logic_vector(3 downto 0);

begin

-- 7 Segment LED Select --
u_led_sel_s : process (clk, rstn)
begin
  if (rstn='0') then
    led_sel_s <= "0111";
  elsif (clk'event and clk='1') then
    led_sel_s <= led_sel_s(2 downto 0) & led_sel_s(3);
  end if;
end process;
```

初期値を“0111”としローテートさせることで，1ビットだけ“L”になる

```
-- 7 Segment LED Select Output Signal --
u_led_sel : process (clk, rstn)
begin
  if (rstn='0') then
    led_sel <= "1111";
  elsif (clk'event and clk='1') then
    led_sel <= led_sel_s;
  end if;
end process;
```

表示パターンの出力に合わせてled_selを出力

```
-- 7 Segment LED Data Output Signal --
u_led_data : process (clk, rstn)
begin
  if (rstn='0') then
    led_data <= "00000000";
  elsif (clk'event and clk='1') then
    if    (led_sel_s(0)='0') then
      led_data <= led1_data;
    elsif (led_sel_s(1)='0') then
      led_data <= led2_data;
    elsif (led_sel_s(2)='0') then
      led_data <= led3_data;
    elsif (led_sel_s(3)='0') then
      led_data <= led4_data;
    else
      led_data <= "00000000";
    end if;
  end if;
end process;

end rtl;
```

表示パターンを切り替えるマルチプレクサ

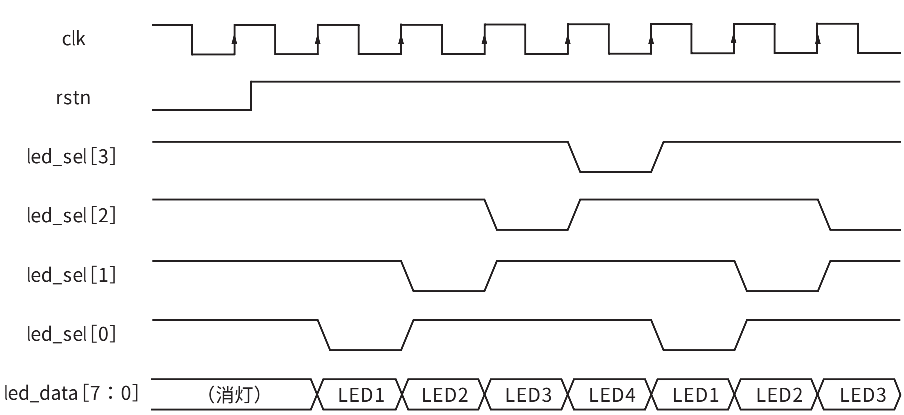


図5-2
ダイナミック点灯のタイミング
led_sel[3：0]は1本ずつ順番に“L”を出力する．led_sel[3：0]を“L”にするタイミングに合わせて，LED表示パターンled_data[7：0]を出力する．

リスト5-2　7セグメント・デコーダのVHDL記述

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity hex2led_decoder is
  Port (
    clk       : in  std_logic;-- Clock
    rstn      : in  std_logic;-- /Reset (Active Low)
    hex_data  : in  std_logic_vector(3 downto 0);
                              -- HEX Data
    led_data  : out std_logic_vector(7 downto 0));
                              -- 7segment Data
end hex2led_decoder;

architecture rtl of hex2led_decoder is

begin

--HEX-to-7segment decoder
-- segment encoding
--      0
--     ---
--  5 |   | 1
--     ---   <- 6
--  4 |   | 2
--     ---   . 7
--      3

u_led_data : process (rstn, clk)
begin
  if (rstn='0') then
    led_data <= "00000000";
  elsif (clk'event and clk='1') then
    case hex_data is
      when "0000" => led_data <= "00111111";  -- Chr 0
      when "0001" => led_data <= "00000110";  -- Chr 1
      when "0010" => led_data <= "01011011";  -- Chr 2
      when "0011" => led_data <= "01001111";  -- Chr 3
      when "0100" => led_data <= "01100110";  -- Chr 4
      when "0101" => led_data <= "01101101";  -- Chr 5
      when "0110" => led_data <= "01111101";  -- Chr 6
      when "0111" => led_data <= "00000111";  -- Chr 7
      when "1000" => led_data <= "01111111";  -- Chr 8
      when "1001" => led_data <= "01101111";  -- Chr 9
      when "1010" => led_data <= "01110111";  -- Chr A
      when "1011" => led_data <= "01111100";  -- Chr b
      when "1100" => led_data <= "00111001";  -- Chr C
      when "1101" => led_data <= "01011110";  -- Chr d
      when "1110" => led_data <= "01111001";  -- Chr E
      when "1111" => led_data <= "01110001";  -- Chr F
      when others => led_data <= "00000000";  -- not light
    end case;
  end if;
end process;

end rtl;
```

7セグメントLEDの表示パターンを記述，点灯するセグメントを‘ 1 ’とする

● 7セグメント・デコーダを追加する

リスト5-1の7セグメントLEDダイナミック点灯ブロックでは，LED表示パターンを入力する仕様になっています．具体的には，led*n*_data［7：0］を，そのままled_dataとして7セグメントLEDへ出力しています．例えば，led1_data［7：0］に0x01を入力すると，LED1ではセグメントaのLEDしか点灯しません．「1」を表示したいときは，「1」の表示パターンである“ 00000110 ”を用意する必要があります．これは数字以外の表示にも利用できる反面，数字表示の目的では使いにくくなります．

0x1を入力したときに「1」と表示（セグメントbとセグメントcを点灯）させるには，バイナリの数値情報を7セグメントLEDの表示パターンに変換するためのデコーダが必要です．**リスト5-2**に，16進数のデータを7セグメントLEDの表示パターンに変換するデコーダを示します．デコーダの記述は簡単です．16進数のデータに対応した表示パターンを出力する回路をcase文で記述します．

図5-3のように，7セグメントLEDダイナミック点灯ブロックledの入力側にこのデコーダhex2led_decoderを追加すると，16進数のデータに対応した7セグメントLEDの表示を行うことができます．

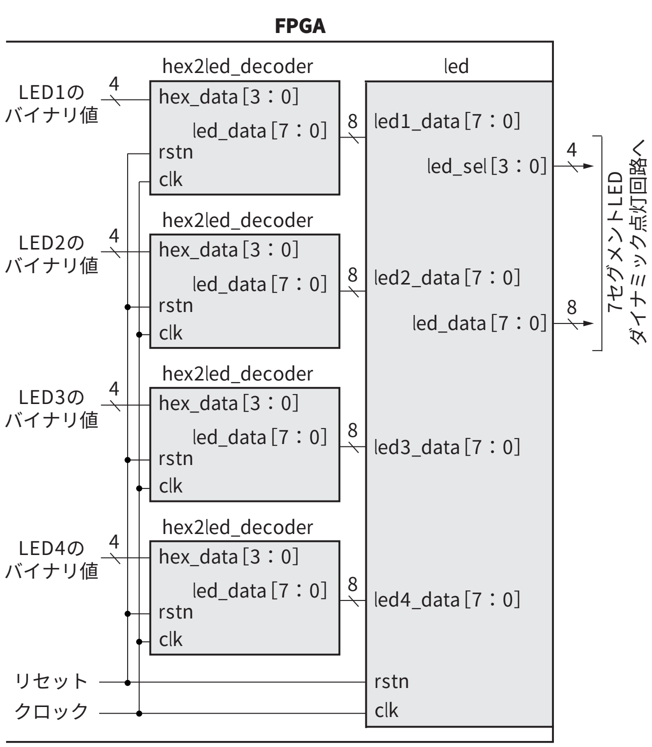


図5-3　7セグメント・デコーダの追加

7セグメントLEDダイナミック点灯ブロックの入力に7セグメント・デコーダを追加すれば，バイナリ値のデータから数字を表示できる．7セグメントLEDダイナミック点灯ブロックと7セグメント・デコーダを完全に独立させたため，4個のデコーダが必要になっている．

まついし・たかひろ
株式会社ダイヘン

06 2色ドット・マトリックスLED点灯回路

河野 崇

外部回路

- 回路図：図6-2
- 主要部品：ドット・マトリックスLED

HDL記述

- VHDL記述：本誌ホームページからダウンロード可能
- 外部入力：なし
- 外部出力：drvRow0，drvRow1，drvCol
- 内部入力：RamData，VRHead，VRTail，VRWrp，CLKDiv，Col2Row，BLCnt，CLK, nRST，nsyncRST
- 内部出力：RamAdrs，RamRdEn，CurCol，Blnk
- パラメータ：COLSIZE（8×8ドット・マトリックスLEDの数），VBDIV，RAMWAIT

評価環境

Quartus II Web Edition 5.0,
Stratix

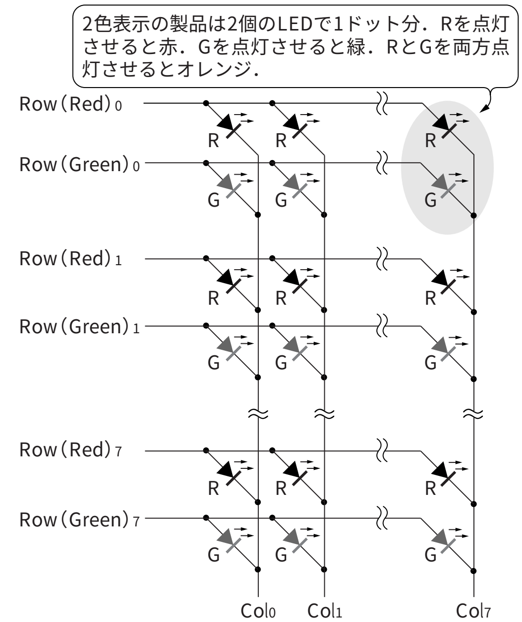


図6-1　ドット・マトリックスLEDの構造

2色表示の例を示す．ドット・マトリックスLEDは多くのメーカから発売されているが，基本的な構造は共通である．単色表示の製品は，1ドットが1個のLEDになる．

7セグメントLEDより多彩な表現ができる部品の一つに，ドット・マトリックスLEDがあります．多数のLEDがマトリックス状に並べられているモジュール部品です．電光掲示板などでよく使われています．

ドット・マトリックスLEDの構造を**図6-1**に示します．

● rowとcolumnを制御して表示する

ドット・マトリックスLEDの表示回路を**図6-2**に示します．

ドット・マトリックスLEDは，2色表示のできる8×8ドットのものを使用できます．FPGAのI/Oでは多数の

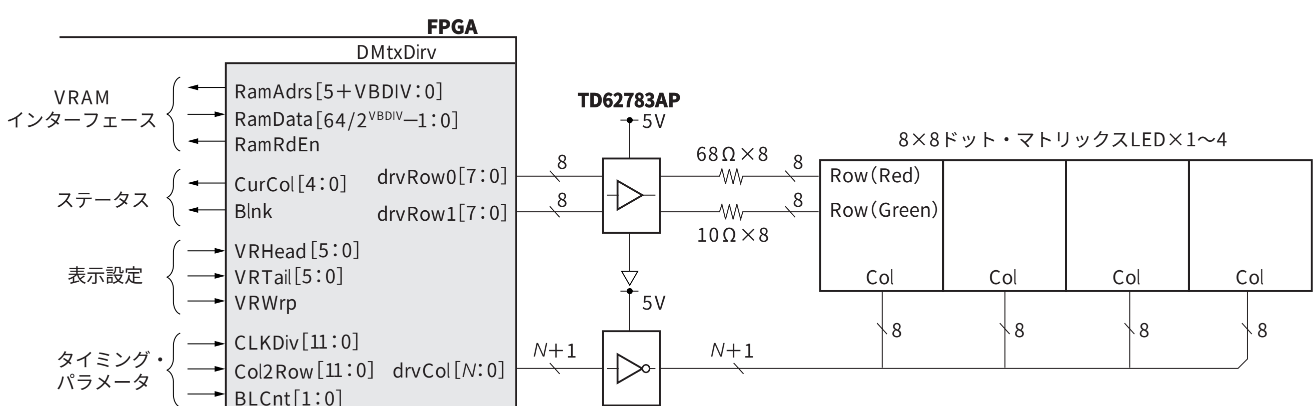


図6-2
ドット・マトリックスLED表示回路の回路図

ドット・マトリックスLEDは，2色表示のできる8×8ドット・マトリックスLEDを1～4個並べて使える．

LEDを直接ドライブできないため，row側（drvRow*X*）にソース・ドライバと制限抵抗，column側（drvCol）にシンク・ドライバを接続しています．ソース・ドライバとしてTD62783AP（東芝），シンク・ドライバとしてTD62083AP（東芝）を使用しました．各ドライバICの電源電圧は5Vにしてあります．LEDには20mAの電流が流れるように，実測値を参考にして抵抗値を決定しました．LEDの電流制限抵抗はドライバの出力飽和電圧とLEDの順方向電圧，電流値をもとにして計算で求めるのが一般的です．しかし，データシートに示された値だけでは，計算で求めたとおりの値にならないこともあります．複数のサンプルによる実測値を参考に決めるほうがよいでしょう．

ドット・マトリックスLED表示ブロックDMtxDrvは，2色を表示できる8×8ドット・マトリックスLEDを1～4個並べて使えるように設計しました．8本×2色分のrowドライブ信号と8本×1～4個のcolumnドライブ信号で

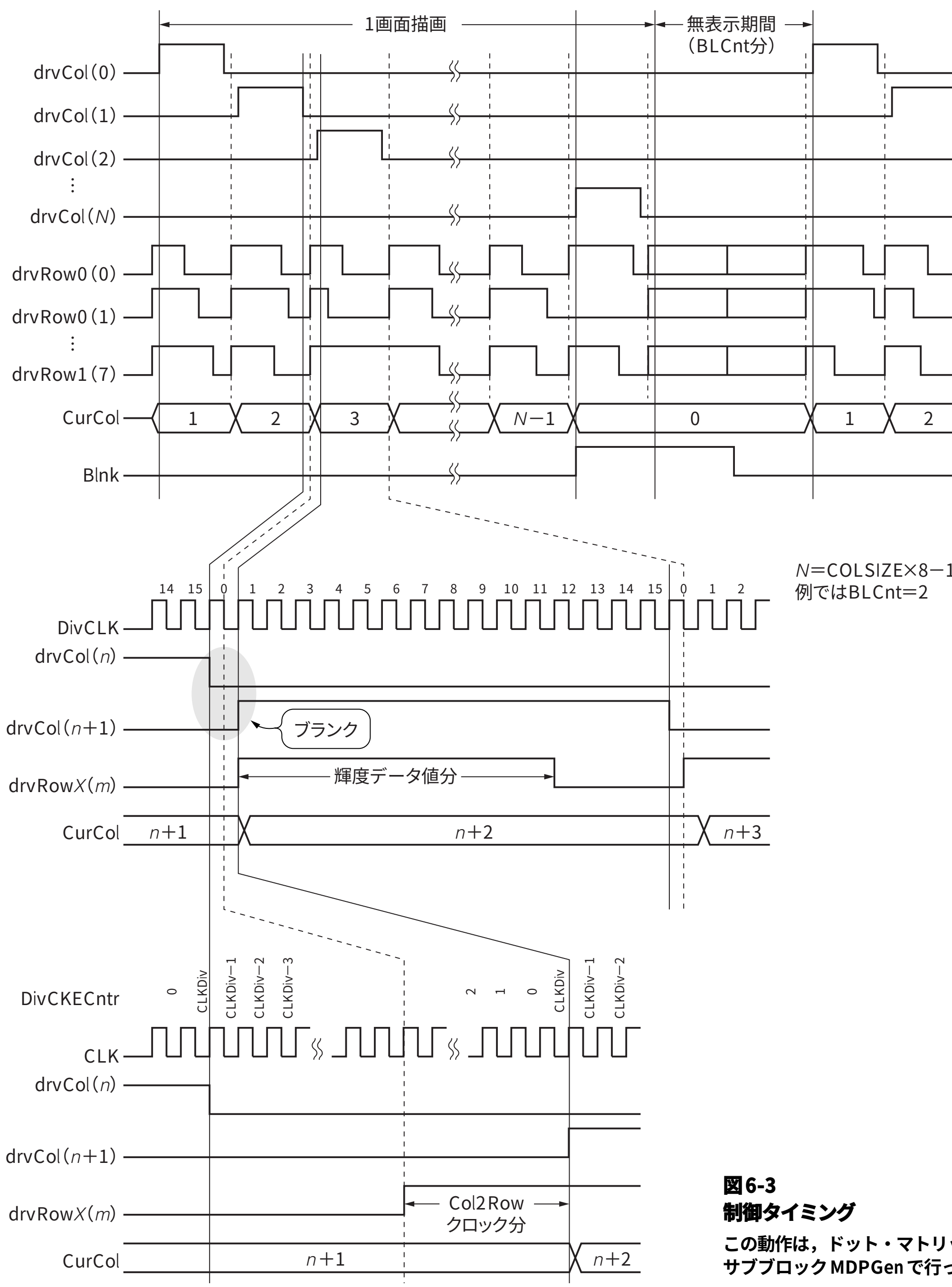


図6-3
制御タイミング
この動作は，ドット・マトリックスLED表示ブロックDMtxDrvのサブブロックMDPGenで行っている．

LEDの点滅を制御します．単色表示の8×8ドット・マトリックスLEDであればrow方向に2個，column方向に1～4個並べることもできます．

● 16階調の明るさを表現する

LEDに表示するパターンは，VRAMに持たせます．VRAMには，1ドット当たり8ビット（1色当たり4ビット）の輝度情報を持たせることにします．

row方向に8ドット，column方向に64ドット，1ドット当たり8ビットの情報なので，8×64×8＝4096ビットの容量が必要です．FPGAの内蔵メモリ・ブロックを使うとちょうどいいようになっています．

LEDの輝度は，点灯時間を変えることによって調整します．点灯時間が長いと明るく，点灯時間が短いと暗く見えるようになります．

制御タイミングを**図6-3**に示します．これはドット・マトリックスLED表示ブロックDMtxDrvのサブブロックMDPGenで行っています．

DivCLKはシステム・クロック（CLK）をCLKDiv分周したクロックで，表示タイミングのベースになっています．1 columnの表示時間は16 DivCLKで，1 DivCLKの間はブランク，15 DivCLKの間は表示という動作を繰り返します．

drvRow*X*から出力する表示パターンの更新はブランクの間に行います．ブランク期間終了の何クロック前に更新するかをCol2Rowで指定します．この値はCLKDivよりも小さくなければなりません．drvRow*X*はdrvColが立ち上がってから（輝度データ値）×（DivCLK周期）の間だけONになります．columnドライブ信号の数はCOLSIZEで指定します（実際の信号数はCOLSIZE×8本）．

1画面描画ごとに何column分の無表示期間をとるかをBLCntで指定します（“00”のときは無表示期間なし）．

● メモリから表示データを読み出す

DMtxDrvから見たVRAMは，64ビット/ワードのリトル・エンディアン構成です（**図6-4**）．LSB側にrow0のデータが入ります．各rowのデータは，ビット3～ビット0がCH0，ビット7～ビット4がCH1の輝度信号です．

VRAMとのインターフェース信号のタイミングを**図6-5**に示します．RamAdrsがアドレス信号で，パラメータのRAMWAITで指定したクロック数（7以下）だけ待ってからRamDataからデータを取り込みます．RamRdEnは出力イネーブル信号で，シングルポート・メモリを用いる場合にアービタに与えるアクセス要求信号として用います．RamDataの幅はパラメータのVBDIVで8，16，32，64から選択することができます．

表示データを読み出すVRAM領域を指定する信号はVRHead，VRTail，VRWrpの三つです．VRHeadとVRTailはそれぞれ表示領域の先頭と後尾のアドレスを示します．VRHeadで指定されるアドレスのデータが左端のcolumnに表示されます．アドレスをインクリメントしながら右側へと表示していきます．右端のcolumnに到達する前にVRAMの後尾に到達した場合はVRAM読み出しアドレスはVRAMの先頭に戻ります．VRTailで指定されるアドレスに到達した場合はVRWrpが‘1’ならば読み出しアドレスがVRHeadに戻り，VRWrpが‘0ならば以降のデ

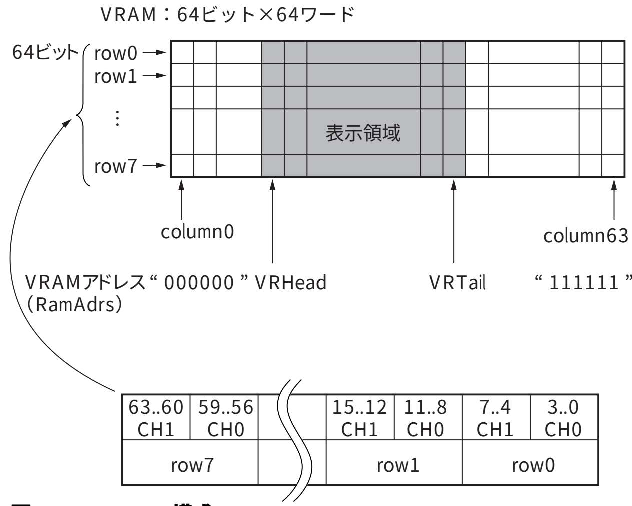


図6-4　VRAMの構成
64ビット/ワードのリトル・エンディアン構成である．LSB側にrow0のデータが入る．

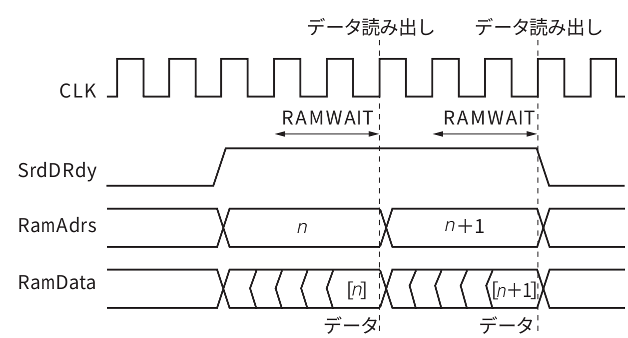
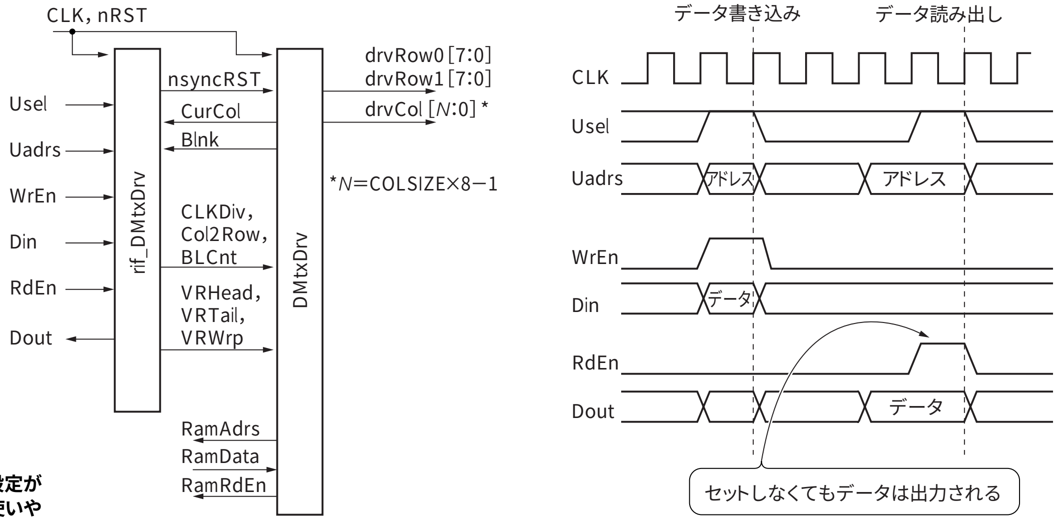


図6-5　VRAMインターフェースのタイミング
アドレスRamAdrsを出力後，RAMWAITで指定したクロック数だけ待ってからデータRamDataを取り込む．



(a) レジスタ・インターフェース

(b) タイミング図

図6-6
レジスタ・インターフェース
レジスタを使ってパラメータ設定ができるようすると，CPUから使いやすくなる．

ータはすべてブランクになります．

Blnk と CurCol はコントローラに現在の描画ステータスを通知する出力信号です．前者は1画面描画後のブランク期間生成時に'1'となり，この期間はVRAMのどの領域に書き込みを行っても画面の乱れが生じないことを保証します．CurColは現在VRAMから読み出しているcolumn番号で，これよりも若いcolumn番号に対応するVRAMアドレスは画面の乱れなしで書き換えることができます．

メモリ制御は，ドット・マトリックスLED表示ブロックDMtxDrvのサブブロックVRamIFで行っています．

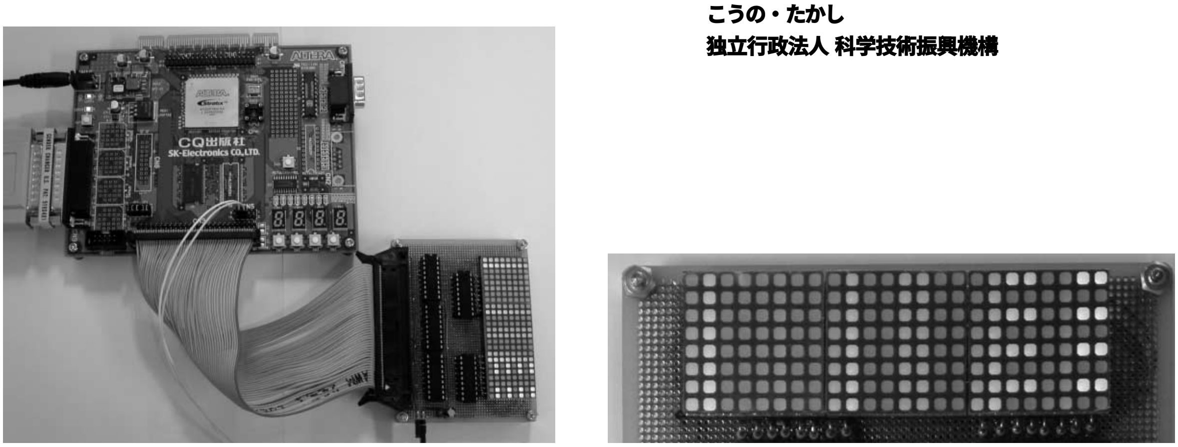
● CPUから使いやすくするレジスタ・インターフェース

CPUなどから使用する場合には，レジスタ・インターフェース（rif_DMtxDrv）があると使いやすくなります．簡略なブロック図を**図6-6(a)**に示します．システム側から供給するCLK，nRST以外の信号はすべてDMtxDrvの同じ名まえの信号に直結します．

レジスタ側の信号は，Uselがセレクト信号，Uadrsがモジュール内アドレス，Dinが入力データ，Doutが出力データ，WrEnが入力データのイネーブル信号，RdEnが出力データの読み出し通知信号です．これらの信号のタイミングを**図6-6(b)**に示します．

動作のようすを**写真6-1**に示します．全ソース・コードは，本誌のホームページからダウンロードできます．また，アップデートは筆者のホームページ（http://www.digicat.info/dwm/）で行う予定です．

こうの・たかし
独立行政法人 科学技術振興機構



(a) ボードとLEDを接続した

(b) ドット・マトリックスLEDの表示

写真6-1　動作のようす

07 アナログ入力回路

相田泰志

外部回路
- 回路図：図7-1
- 主要部品：A-Dコンバータ（MCP3208）

HDL記述
- VHDL記述：リスト7-1
- 外部入力：ad_dout
- 外部出力：ad_clk，ad_cs，ad_din
- 内部入力：clkin，rst
- 内部出力：rbuf
- パラメータ：なし

評価環境

ISE WebPACK 7.1i，Quartus II Web Edition 5.0，自作評価ボード

FPGAだけではどうしてもできないことの一つに，アナログ・インターフェースがあります．1チップ・マイコンはA-Dコンバータが内蔵されているものがほとんどですが，FPGAでA-Dコンバータを内蔵する製品はめったにありません．

● センサや電圧の入力を行う回路

図7-1に逐次比較型A-Dコンバータによるアナログ入力回路を示します．米国Microchip Technology社の「MCP3208」は，12ビット精度で8チャネルの入力を持っています．電源電圧2.7V～5.5Vで動作するため，FPGAのI/O電源が3.3Vであれば直接接続できます．変換速度は，2.7V動作時で最大50kサンプル/sです．ただし，サンプリング・コンデンサの制限によって，サンプリング時間は1.2ms程度必要です．変換されたデータはシリアル・インターフェースで出力されます．動作タイミングを**図7-2**に示します．

$\overline{CS}$を“L”にした後，クロック（CLK）に合わせて変換スタート・ビットや変換するチャネル番号を送信します．すると，D_{OUT}から変換されたデータが上位ビットから順に出力されます．

チャネル1のデータを読み取る例を**リスト7-1**に示します．内部ノードのsbufにチャネル1を指定するパラメータをあらかじめセットしておきます．これを上位ビットから送信し続けます．同時にD_{OUT}のデータを読み取り，sbuf［0］に保存していきます．20クロック後にはsbuf［11：0］にディジタル値が収められます．

なお，ここではエラー処理などは行っていません．

● 温度データを読み取る

図7-1の回路では，0～3.3Vの電圧を入力することができます．しかし，センサなどから出力される電圧は，0～3.3Vであるとは限りません．そこで入力される信号をA-D変換に適した信号にするための回路（プリアンプ）が必要になります．

例えば気温や水温など，0℃～120℃ぐらいの温度を入力するのに便利なセンサとしてサーミスタがあります．サーミスタは周囲温度によって抵抗値が変化する特性を持ち

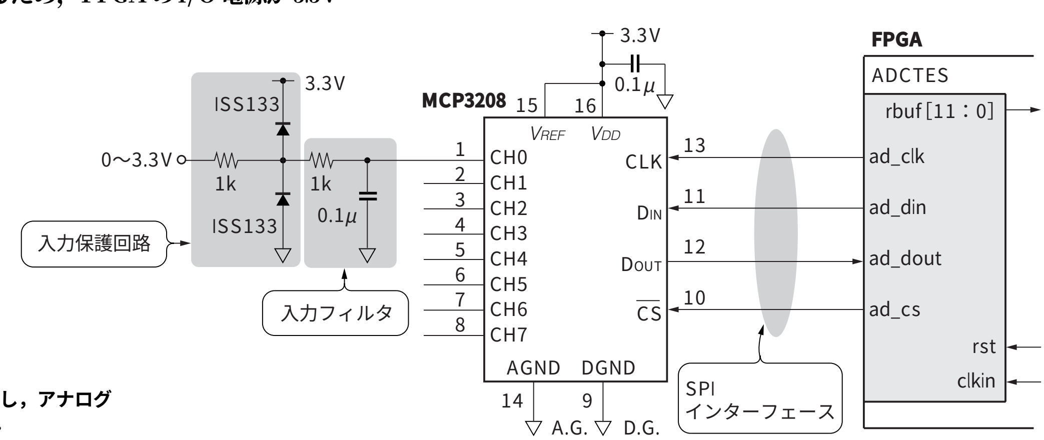
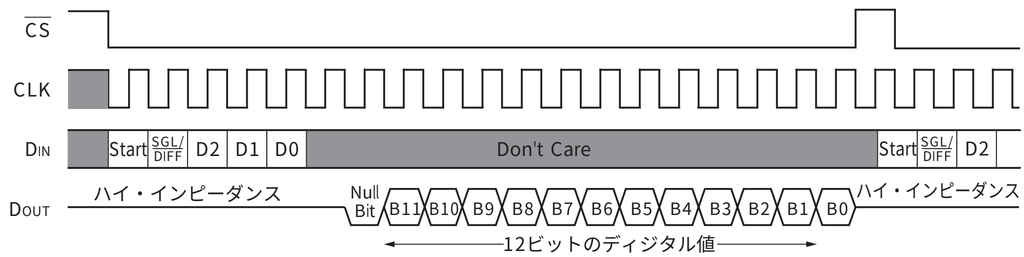


図7-1 アナログ入力回路
逐次比較型A-Dコンバータを利用し，アナログ信号をディジタル信号に変換する．

図7-2
MCP3208の動作タイミング
スタート・ビット(Start)やアナログ信号のタイプ(SGL/DIFF),アナログ入力チャネル(D2～D0)を指定すると,12ビットのディジタル値(B11～B0)が送られてくる.



リスト7-1　チャネル1のアナログ値を読み取るVHDL記述

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all ;
use ieee.std_logic_arith.all ;

entity ADCTES is port (
      clkin     : in std_logic ;   -- CLK IN
      rst       : in std_logic ;   -- reset
      ad_clk    : out std_logic ;  -- ADC clock out
      ad_cs     : out std_logic ;  -- ADC cs
      ad_dout   : in std_logic ;   -- ADC data out
      ad_din    : out std_logic ;  -- ADC data in
      rbuf      : out std_logic_vector( 11 downto 0)
      ) ;
end ADCTES ;

architecture arch_ADCTES of ADCTES is

signal sbuf     : std_logic_vector( 19 downto 0) ;
                                       -- Send Buffer
signal wcnt     : std_logic_vector(4 downto 0);

begin

ad_clk <= clkin;

process (clkin, rst)
begin
  if rst = '0' then
    ad_cs <= '1' ;
    sbuf <= "01100100000000000000"; -- set ADC CH.1
    wcnt <= "00000" ;
  elsif ( clkin'event and clkin = '1' ) then
    wcnt <= wcnt + 1;
    if (wcnt = "10100") then
      wcnt <= "00000";
      rbuf(11 downto 0) <= sbuf(11 downto 0);
      ad_cs <= '1' ;
      sbuf <= "01100100000000000000";-- set ADC CH.1

    else
      ad_cs <= '0' ;
      sbuf(19 downto 1) <= sbuf( 18 downto 0);
                                       -- send Command
      ad_din <= sbuf(19);
      sbuf(0) <= ad_dout;
    end if ;
  end if ;
end process ;

end arch_ADCTES;
```

ます.抵抗値を読み取れば温度を検出することができるというわけです.

図7-3にサーミスタのインターフェース回路を示します.サーミスタと抵抗によって分圧した電圧を取り込むことにより,温度を電圧に変換しています.検出したい温度のサーミスタの抵抗とサーミスタ用分圧抵抗を等しくすると,

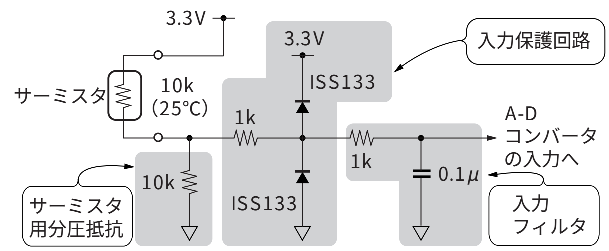


図7-3　サーミスタ入力回路
温度センサの一つであるサーミスタを使うための回路である.

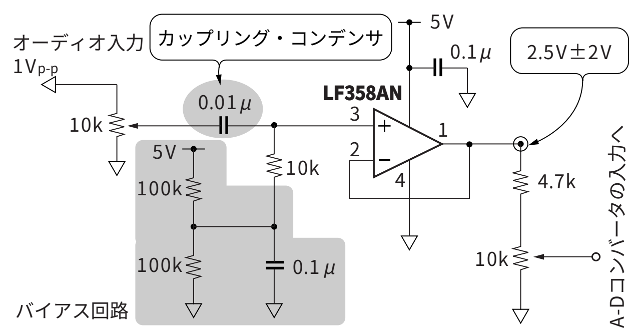


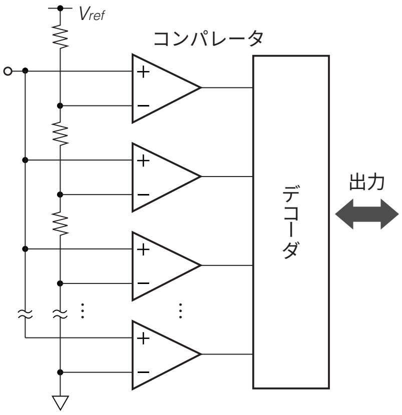
図7-4　音声入力回路
オーディオ機器で使われる音声信号を読み取るための回路である.

精度が高くなります.

● オーディオ信号を入力する

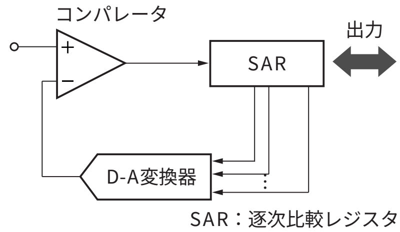
オーディオでは,音声信号をRCAピン・ジャックなどで接続します.ここで使われている信号は,約$1V_{p\text{-}p}$,20 kHz程度までの低周波信号です.

このような低周波信号を入力するときに使う回路を**図7-4**に示します.音声信号はAC信号であるため,入力をコンデンサでカップリングして入力します.使用しているOPアンプ「LF358」は5V単一電源で動作するため,信号がなにも入っていないときの出力値が2.5Vとなるようにバイアスを加えます.そして最終的な出力がA-Dコンバータの入力である0～3.3Vに対応するように,抵抗で分圧します.結果的に,無信号のときには1.15Vになります.



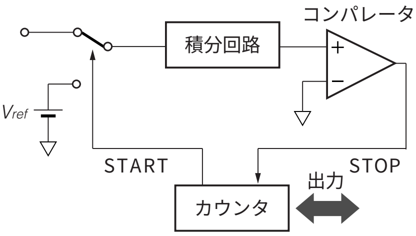
特　徴　：分解能分のコンパレータ（比較器）を並べ，入力電圧を比較する方式．高速な変換が可能．NビットのA-Dコンバータであれば，2^N個のコンパレータが必要なため，高分解能なものは，製造が困難．
分解能　：8〜10ビット
サンプリング速度：GHzのサンプリングも可能
おもな用途：高周波，ビデオ信号などの信号処理

（**a**）フラッシュ型



特　徴　：D-A変換器から出力した値をコンパレータと比較しながら変換する方式．上位ビットから下位ビットまで順次比較していく．マイコン内蔵型のA-Dコンバータはこの方式が多い．
分解能　：8〜14ビット
サンプリング速度：10k〜数MHz程度
おもな用途：計測器など広範囲

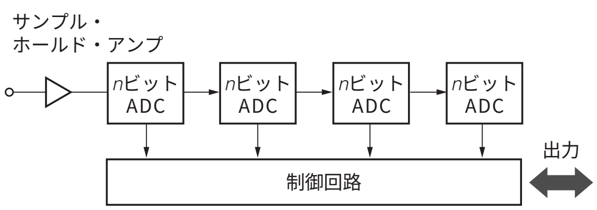
（**b**）逐次比較型



特　徴　：積分器（コンデンサ）で入力電圧を積算していき，その積算時間でA-D変換を行う方式．高精度だが低速．二重積分型は変換前に基準の電圧で積分することによって，オフセットやノイズなどの影響を避けるようにした方式．積分時間を長くすればするほど，精度を上げることができる．アナログ部分が多いため，製造が難しい．
分解能　：12〜20ビット
サンプリング速度：1kHz以下
おもな用途：計測器など高精度を要求する分野

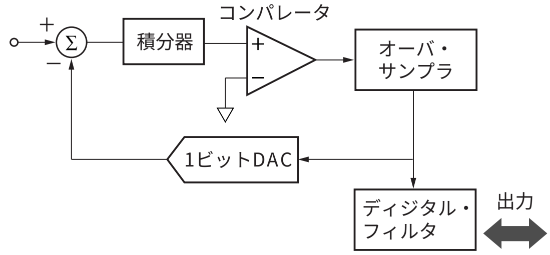
（**c**）二重積分型

図7-5
A-D変換方式
代表的な方式のみを示した．ここに示したもの以外にもさまざまな方式が提案されている．



特　徴　：各段ごとの分解能を少なくし，パイプライン処理のように，順にA-D変換を行う方式．段数を上げることで高速なまま分解能を上げることができる入力から出力までの遅れは発生するが，サンプリング速度，分解能を上げることができる．高周波の応用などで注目されている．
分解能　：10〜20ビット
サンプリング速度：最大80MHz
おもな用途：通信機器

（**e**）Σ-Δ型



特　徴　：コンパレータでアップ・ダウン・カウンタを動かし，追従させていくデルタ・コンバータを改良した方式．オーバ・サンプラとディジタル・フィルタによって，サンプリングを速くしたのと同等の効果を得ることができる．高精度ほとんどがディジタル回路のため，LSI化しやすく安価.連続して変換していく必要があるため，多チャネルを切り替える用途には適用できない．
分解能　：12〜24ビット
サンプリング速度：100kHz以下
おもな用途：計測からオーディオまで幅広い

（**d**）パイプライン方式

MP3208ではサンプリング速度が遅いため，音声帯域全体の信号をとることはできません．音声帯域全体をとるのであれば，サンプリング定理により40kHz以上のサンプリングが必要になります．また，正確な収集を行うためにはサンプリング速度以上の信号が影響するため，アンチエイリアシング・フィルタを設ける必要があります．

● 最適なA-Dコンバータを選ぶ

A-Dコンバータには，**図7-5**に示すような変換方式があります．対象とする信号に合わせて，最適な変換方式のA-Dコンバータを選択します．

また，A-DコンバータとFPGAやCPUなどとのインターフェースにもいくつかの方式があります．例えば，パラレル・インターフェースは，レジスタを直接読み込めるため，高速サンプリングに向いています．また，FPGAによる処理も簡単です．ただし，接続する線が多くなるため，パッケージが大きくなり，実装面積も大きくなりがちです．パラレルといっても，高速動作のために差動信号が使われることもあります．

シリアル・インターフェースとしては，SPI，I^2Cなどの方式を利用したものが多くみられます．高速なデータ伝送は行えませんが，省スペースで接続できます．

そうだ・やすし　西山工業（株）

08 アナログ出力回路

相田泰志

外部回路
- 回路図：図8-2
- 主要部品：OPアンプ（LM358AN）

HDL記述
- VHDL記述：リスト8-1
- 外部入力：なし
- 外部出力：pwm_out
- 内部入力：clkin，ccnt
- 内部出力：なし
- パラメータ：なし

評価環境

ISE WebPACK 7.1i，Quartus II Web Edition 5.0，自作評価ボード

ディジタル・データからアナログ信号を出力するときには，D-A変換を行います．D-A変換には，おもに**図8-1**のような方式があります．

図8-1（a）の*R*-2*R*抵抗方式がいちばんよく用いられています．ビットを増やすと精度も上がります．この方式を用いたD-Aコンバータも発売されています．

図8-1（b）に示すPWM方式は，外部回路の構成が簡単です．**図8-2**にPWM方式を使用したアナログ出力回路を示します．これはディジタル出力ポートにフィルタがあるだけの回路です．

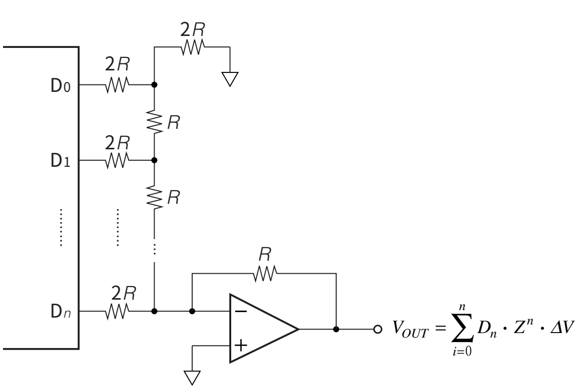
図8-3のように，デューティ比を変化させたパルスを出力すると，フィルタにより平滑化されたアナログ信号になります．つまり，デューティ比に対応した電圧を出力することができるようになるわけです．ただし，フィルタの周波数に対して，十分に周期の短いパルスを出力する必要があります．

PWM方式によりアナログ値を出力するVHDL記述を**リスト8-1**に示します．tcntが0～999のカウントでフリーに動作しています．ccntにディジタル値（0～999）を設定します．tcnt＜ccntのときに‘1’，ccntを超えると‘0’を出力します．この動作によって，出力電圧

$$V_{OUT} = \text{ccnt}/1000 \times V_{cc}$$

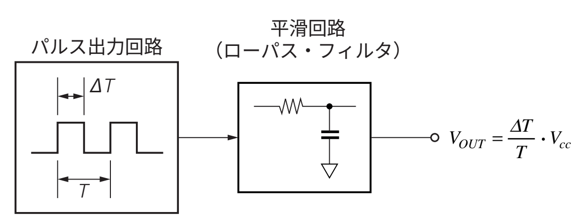
のアナログ信号を出力することができます．

*CR*フィルタによるアナログ出力の場合，フィルタの特性により，ゲインが下がり，正確な直線性が得られない場合があります．その際には，アクティブ・フィルタなどの*Q*が高く，透過ゲインが一定なフィルタを用いることによ



特　徴　　：抵抗によって電流を加算するしくみを利用する方式．もっとも一般的で，低速から高速まで利用される．
分解能　　：8～48ビット
おもな用途：高速オーディオ，ビデオ，そのほかの産業用など

（a）*R*-2*R*抵抗方式



特　徴　　：パルス信号をフィルタにより平滑化することによって電圧を出力する方式．パルスのデューティ比を変化させることで出力電圧を変化させる．フィルタだけで構成できるため，回路が簡単．フォトカプラなどによる絶縁が簡単に行える．低速．
分解能　　：―
おもな用途：低速産業用など

（b）PWM方式

図8-1　D-A変換方式

***R*-2*R*抵抗方式はもっともよく用いられている．PWM方式は外部回路が簡単になるという特徴がある．**

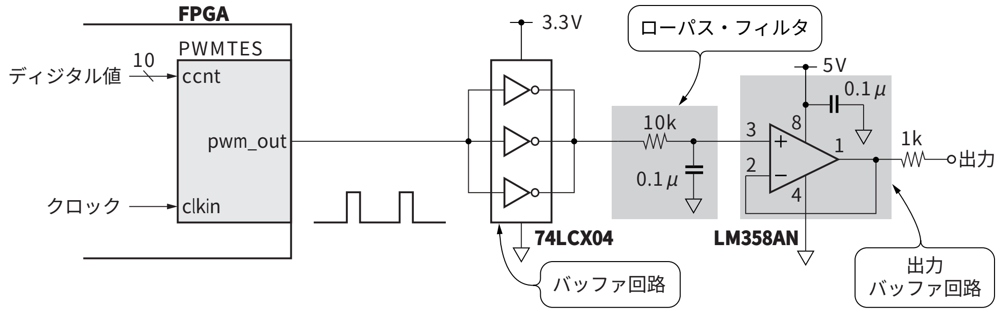


図8-2
PWM方式によるアナログ出力回路
ディジタル出力ポートにフィルタがあるだけの回路である．

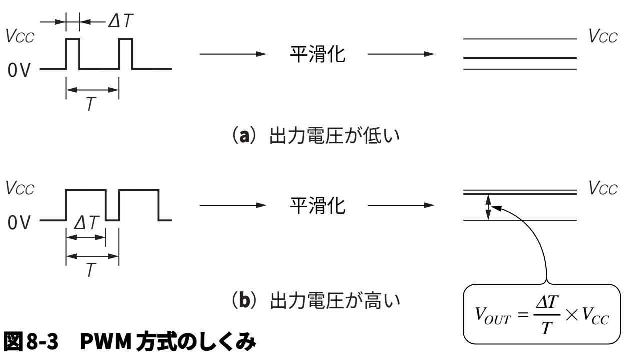


図8-3　PWM方式のしくみ
デューティ比を変化させたパルスを出力すると，フィルタにより平滑化されたアナログ信号になる．

り，より精度が高い出力回路を構成することができます．アクティブ・ローパス・フィルタの例を**図8-4**に示します．

またカウンタの最大値を大きくすることで，精度を上げられます．しかし，それでは1サイクルの出力時間が長くなってしまいます．クロックを上げる，あるいはフィルタを大きなものにするなどの配慮が必要です．

そうだ・やすし
西山工業(株)

リスト8-1　PWM方式によりアナログ値を出力するVHDL記述

```
library IEEE;
use IEEE.std_logic_1164.all ;
use IEEE.std_logic_unsigned.all ;
use ieee.std_logic_arith.all ;

entity PWMTES is
port (
    clkin      : in std_logic ;   -- CLK IN
    ccnt       : in std_logic_vector(9 downto 0);
                                  -- 10bit Capture Counter
    pwm_out    : out std_logic    -- PWM OUT
    ) ;
end PWMTES ;

architecture arch_PWMTES of PWMTES is

signal tcnt    : std_logic_vector(9 downto 0);
                                     -- 10bit Timer Counter

begin

process(clkin,ccnt)
begin
    if (clkin'event and clkin='1') then
        if (tcnt < 999 ) then
                                      -- 0-999 loop counter
            tcnt <= tcnt+1;
        else
            tcnt <= "0000000000";
        end if;
        if (tcnt < ccnt ) then
            pwm_out <='1';
        else
            pwm_out <='0';
        end if;
    end if;
end process;

end arch_PWMTES;
```

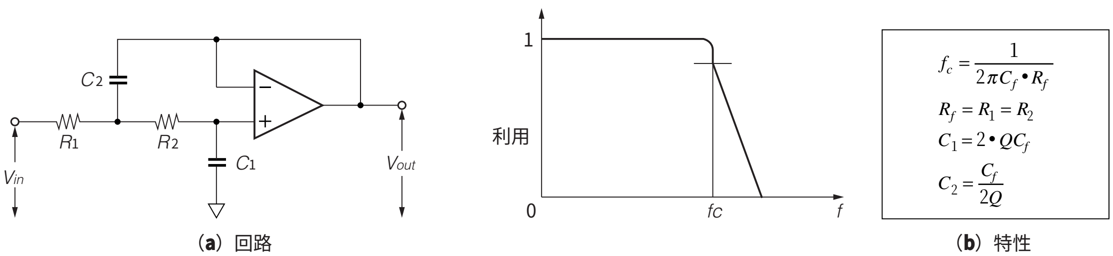


図8-4　アクティブ・ローパス・フィルタ
***CR*フィルタによるアナログ出力では，正確な直線性が得られない．アクティブ・フィルタを用いると，より精度が高い出力回路を構成できる．**

09 高速アナログ出力回路

田中良平

外部回路
- 回路図：図9-1
- 主要部品：D-A コンバータ（DAC902）

HDL記述
- VHDL 記述：リスト9-1
- 外部入力：なし
- 外部出力：dac，dacclk，pd
- 内部入力：dout，clk，reset
- 内部出力：なし
- パラメータ：なし

評価環境

ISE WebPACK 7.1i，Spartan-II E300 評価キット

高速なサンプリング周波数が必要なときは，パラレル・インターフェースを持つD-A コンバータを利用します．

パラレル・インターフェースの制御回路は非常にシンプルです．しかし，パラレルであるがゆえにピン数の増大を避けることができません．多チャネルが必要な場合，接続するFPGA も多ピン・パッケージを選択することになるので，コストアップにつながります．ほんとうに高速なD-A コンバータが必要なのかどうかを見極めることがたいせつです．

● 高速サンプリングの回路ではパターン設計もたいせつ

パラレル・インターフェースを持つ高速D-A コンバータを使ったアナログ出力回路を**図9-1** に示します．米国Texas Instruments 社の12 ビットD-A コンバータ「DAC

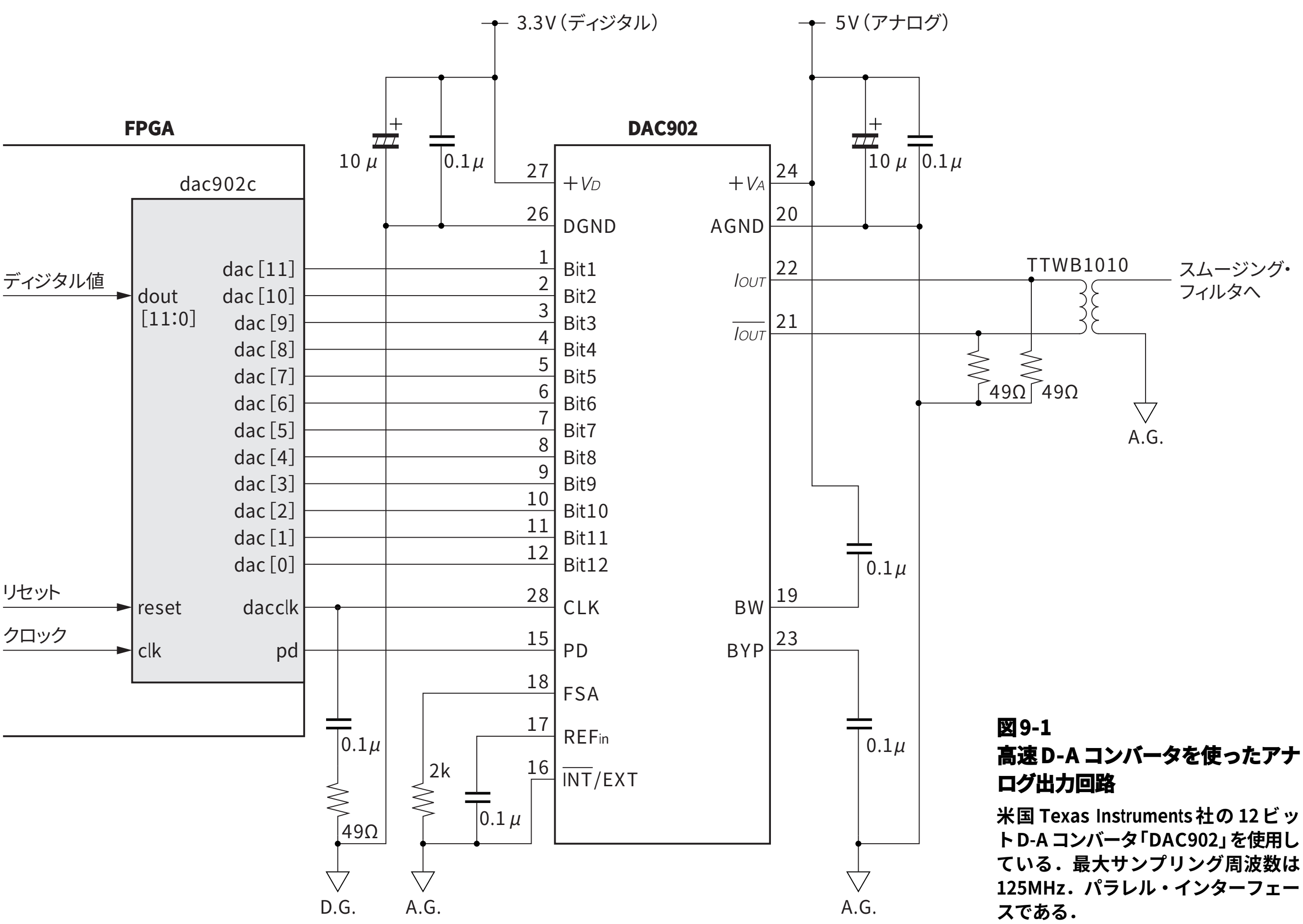
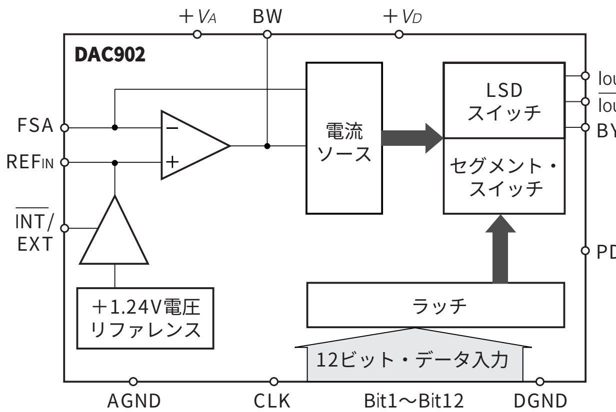
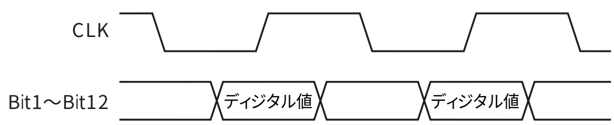


図9-1
高速D-A コンバータを使ったアナログ出力回路
米国Texas Instruments 社の12 ビットD-A コンバータ「DAC902」を使用している．最大サンプリング周波数は125MHz．パラレル・インターフェースである．



(a) ブロック図



(b) タイミング・チャート

図9-2 DAC902の概要

クロックの立ち上がりでディジタル値を読み込み，D-A変換してアナログ出力する．非常にシンプルな動作である．ディジタル値はBit1がMSB，Bit12がLSBである．

902」を使用しています．最大サンプリング周波数は3.3V動作時が125MHz，5V動作時が165MHzです．

当然のことですが，高速D-Aコンバータを使用する場合は，サンプリング・クロックが高速になります．ディジタル側のクロック波形がアナログ出力にノイズとして影響しないようなくふうが必要になります．

例えば，D-Aコンバータへ入力するクロック信号はできるだけ短くします．どうしても長くなってしまうときには，クロック信号をGNDでシールドする必要性がでてきます．データ・ピンはFPGAと1対1で接続するだけですが，本数が多いので，効率良くパターンを引く必要があります．

● シンプルな制御ブロック

パラレル・インターフェースのD-Aコンバータは，制御が非常にシンプルです．タイミング・チャートを**図9-2**に，ブロック図を**図9-3**に，VHDL記述を**リスト9-1**に示します．

正弦波のような符号付きデータを出力する場合は，バイナリ・オフセット出力にし，最上位ビットだけを反転して出力します．タイミングは，サンプリング・クロックの立ち上がりでD-Aコンバータがデータをラッチできるようにします．

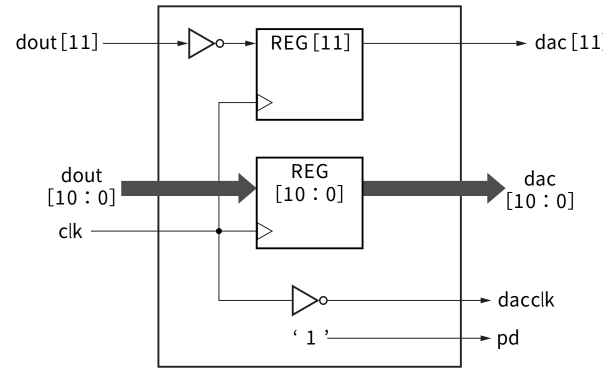


図9-3 DAC902の制御ブロック図

バイナリ・オフセット出力にする．最上位ビットだけを反転して出力する．

リスト9-1 高速D-Aコンバータを制御するVHDL記述

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity dac902c is
    Port ( clk : in std_logic;
           reset : in std_logic;
           dout : in std_logic_vector(11 downto 0);
           dac : out std_logic_vector(11 downto 0);
           dacclk : out std_logic;
           pd : out std_logic);
end dac902c;

architecture rtl of dac902c is

begin

u_dac_reg :process(clk,reset)begin
    if(reset = '1')then
        dac <= "100000000000";
    elsif(clk'event and clk = '1')then
        dac <= not dout(11) & dout(10 downto 0);
    end if;
end process;

u_dacclk : dacclk <= not clk;

pd <= '1';

end rtl;
```

たなか・りょうへい
(株)ダイヘン

2

10 4チャネル・アナログ出力回路

田中良平

外部回路
- 回路図：図10-1
- 主要部品：D-A コンバータ（DAC7615E）

HDL記述
- VHDL 記述：本誌ホームページからダウンロード可能
- 外部入力：なし
- 外部出力：SDI，CLKn，CSn，LOADREGn，LOADDACS，RESETn，RESETSEL
- 内部入力：ADC1，ADC2，ADC3，ADC4，RESET，CLK
- 内部出力：なし
- パラメータ：なし

評価環境

ISE WebPACK 7.1i，Spartan-II E300

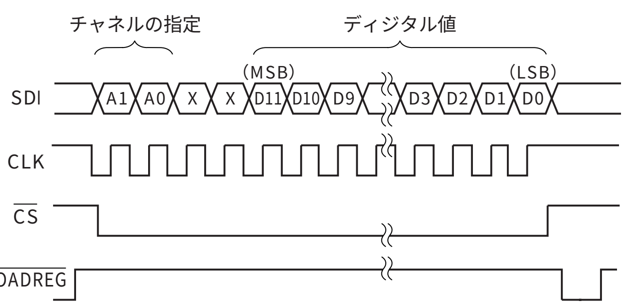


図10-2　DAC7615E のタイミング・チャート
設定するチャネルとディジタル値をシリアルで送信する．

サンプリング周波数が低くてよく，複数のアナログ出力が必要な場合には，シリアル・インターフェースのD-Aコンバータがよく利用されます．

ディジタル側がシリアル・インターフェースになると，制御回路が複雑になる傾向があります．しかし，チャネル数が増えてもピン数が少なくてすみます．

● 複数のチャネルでも少ない信号で制御できる

シリアル・インターフェースを持つ多チャネルD-Aコンバータを使ったアナログ出力回路を**図10-1**に示します．米国Texas Instruments社の4チャネル12ビットD-Aコンバータ「DAC7615E」を使用しています．最大サンプリング周波数は120kHzです．

ディジタル側はFPGAと1対1で接続するだけです．

アナログ出力を0～2.5Vの範囲で変化させるため，V_{REFL}をGND接続，V_{REFH}を2.5Vに接続しています．2.5Vは安定したレベルを確保する必要があるため，Texas Instruments社の電圧リファレンス「REF3125」を使用して生成しています．V_{REFH}に入力できる最大値は2.5Vです．

アナログ出力には，必要に応じてローパス・フィルタを接続します．

● データ・シフトとデータ・セットを繰り返す

D-AコンバータとFPGAの間はシリアル・インターフェースです．タイミング・チャートを**図10-2**に示します．

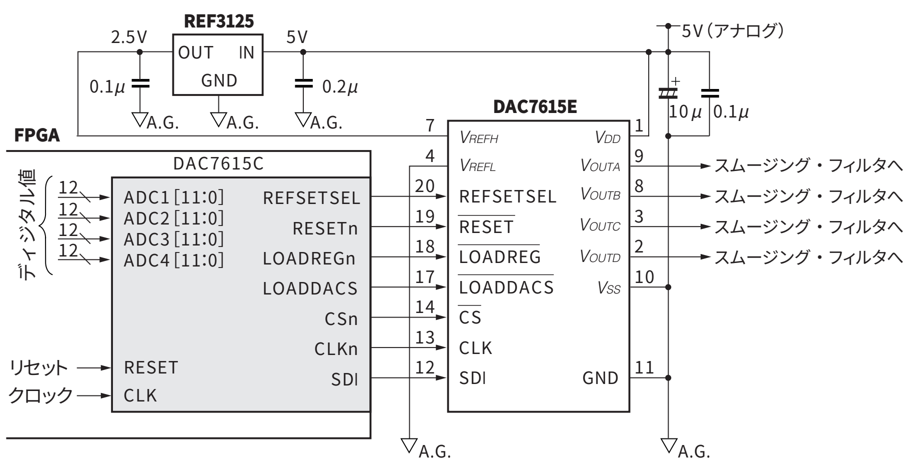


図10-1
4チャネル・アナログ出力回路の回路図
米国Texas Instruments社の4チャネル12ビットD-Aコンバータ「DAC7615E」を使用している．最大サンプリング周波数は120kHz．

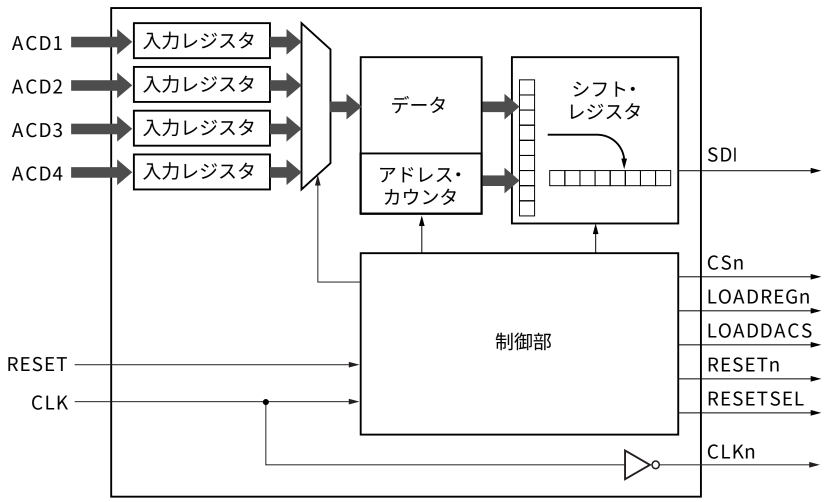


図10-3　DAC7615 インターフェース・ブロック
入力レジスタに設定されたディジタル値をシリアル変換して出力する．

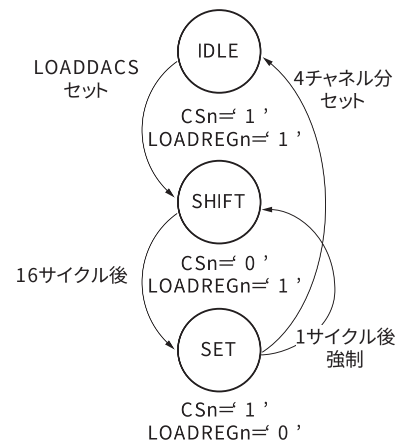


図10-4　DAC7615 インターフェース・ブロックの制御部の状態遷移図
データ・シフトとデータ・セットを4回繰り返す．

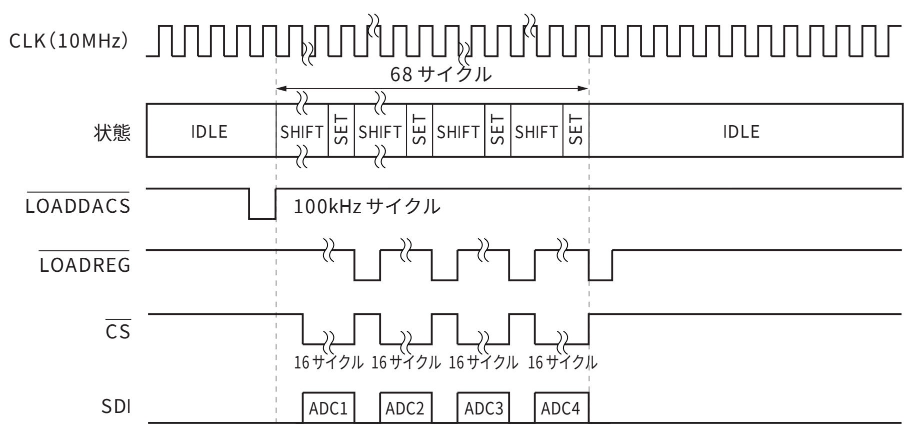


図10-5
データ設定シーケンス
4チャネル分のデータを送信するためには，最低で68サイクル必要．

D-A コンバータの内部には，4チャネル分のD-A変換器があり，それぞれに対してディジタル値を設定できます．そこで，設定するチャネルとディジタル値をシリアルで送信します．

すべてのチャネルからアナログ信号を出力するブロック図を**図10-3**に示します．

制御部は，**図10-4**に示すステート・マシンで構成されています．データ・シフトとデータ・セットを4回繰り返すことですべてのチャネルの設定が終わります．アイドル状態も含めて3状態の遷移となります．

四つのチャネルの入力レジスタに保持されたディジタル値を順番に読み込み，アドレス情報を付加し，パラレル-シリアル変換してD-Aコンバータへ送信します．

D-Aコンバータのアナログ出力は，LOADDACS信号によって更新されます．そこで，この周期内で全4チャネルのデータ・セットを完了する必要があります．4チャネル分のデータを送信するためには，最低でも68サイクル必要です(**図10-5**)．ここでは簡単に，10MHzのシステム・クロックを使うときは，100kHzサンプリング周期で更新するように，100サイクルごとに$\overline{\text{LOADDACS}}$信号をアクティブにしています．

RESETSEL信号を'1'にすると，リセット時のD-Aコンバータのアナログ出力はh800を変換した出力となります．**図10-1**の回路であれば1.25Vが出力されます．リセット時に0Vとするには，RESETSEL信号を'0'にします．

たなか・りょうへい
(株)ダイヘン

11 同期SRAMインターフェース回路

吉田政和

外部回路

- 回路図：図11-1
- 主要部品：ZBT-SSRAM（GS8320Z36T-166）

HDL記述

- Verilog HDL記述：本誌ホームページからダウンロード可能
- 外部入力：ssram_q
- 外部出力：ab，csn，wn，gn，bwan，bwbn，bwcn，bwdn，adv，cken
- 内部入力：clk，start，lfsr，rstn
- 内部出力：write_enable
- パラメータ：MAIN_IDLE，MAIN_WRITE，MAIN_READ，MAIN_COMP_WAIT，MAIN_COMP，MAIN_INC，MAIN_BAD，WRITE_IDLE，WRITE_CMD，WRITE_WAIT1，WRITE_WAIT2，WRITE_WAIT3，WRITE_WRITE1，WRITE_WRITE2，WRITE_WRITE3，WRITE_WRITE4，READ_IDLE，READ_CMD，READ_WAIT1，READ_WAIT2，READ_WAIT3，READ_WAIT4，READ_READ1，READ_READ2，READ_READ3，READ_READ4

評価環境

Quartus II Web Edition 5.0，Stratix II

ZBT-SSRAMは，一般的な同期型SRAMのパイプライン・バースト・モードに加え，レイテンシが1少ないNBT（No Bus Turn Around）モードを持ちます．コストよりも性能を追求するような特定のアプリケーションでよく使われています．モードはFTnピンで切り替えます．

各モードの場合のレイテンシは，以下のとおりです．

- パイプライン・バースト・モード：3-1-1-1
- NBTバースト・モード：2-1-1-1

ただし，NBTバースト・モードでは，パイプライン・バースト・モードの最大クロックより2～4割ほど周波数を落とさなくてはなりません．GS8320Z36T-166をNBTバースト・モードで動作させると，システム・クロック周波数は125MHzが上限です．

● パイプライン・バースト・モードで動作させる

ZBT-SSRMインターフェース回路を**図11-1**に示します．ダンピング抵抗が必要ですが，ほとんど直結ですみます．

ZBT-SSRAM制御ブロックは，メイン，ライト，リード

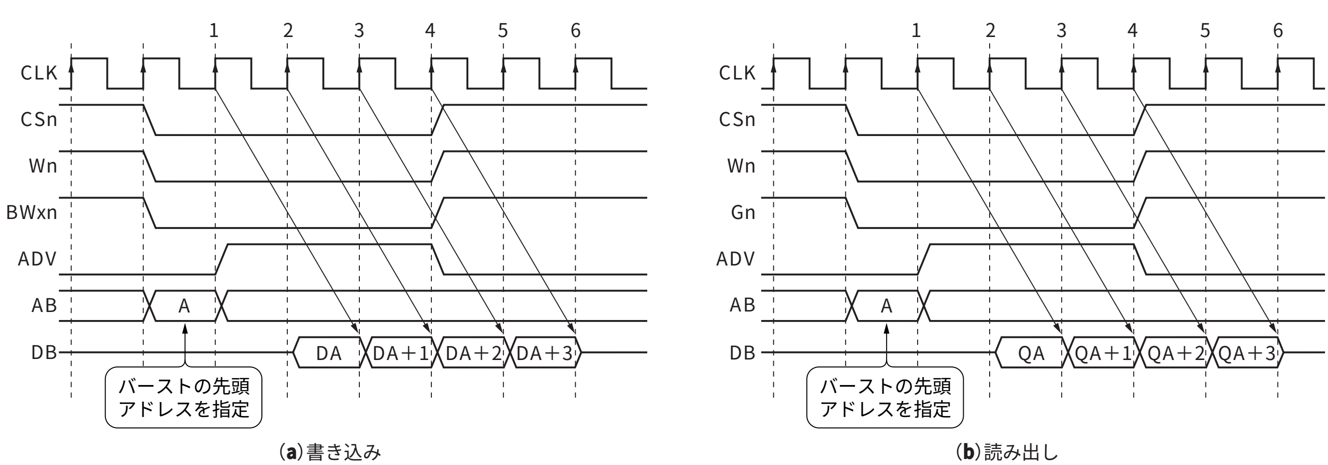


図11-2　パイプライン・バースト・モードのタイミング・チャート

4バイト・リニア・バースト・モードの動作を示す．図に記載されていない信号のうち，CKEとFTnは“H”，ZZとLBOnは“L”である．

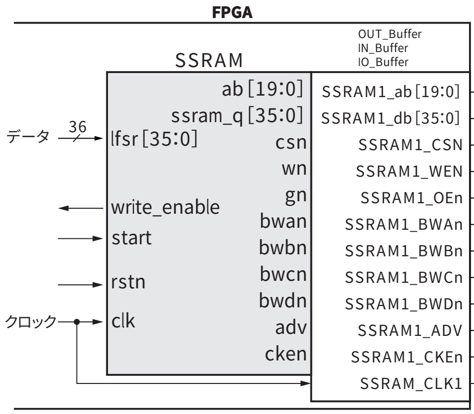
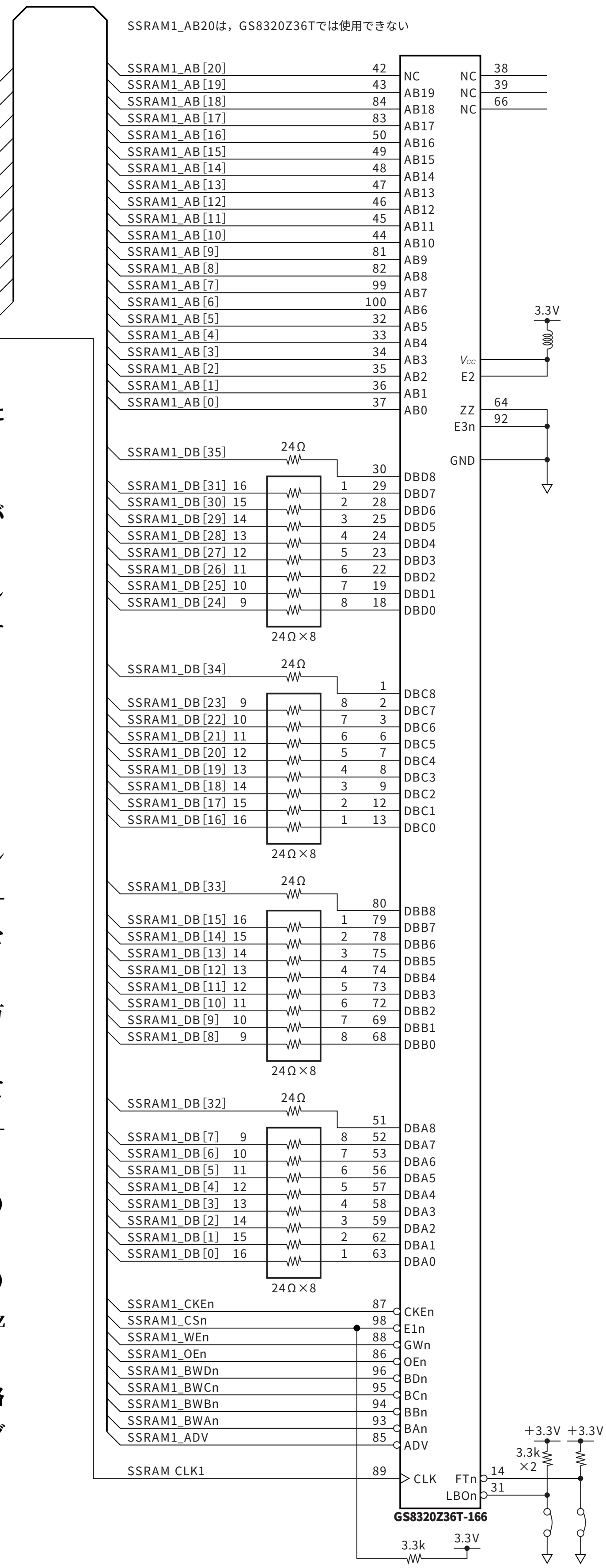


図11-1 ZBT-SSRMインターフェース回路
ダンピング抵抗が必要だが，ほとんど直結でよい． SSRAMの外部入出力には，I/Oバッファを入れる．



という3個のステート・マシンと4×36ビットのリード・バッファ，4×36ビットのライト・バッファを持ちます．

ZBT-SSRAMがバースト動作を行うように信号を生成します．パイプライン・バースト・モードのタイミング・チャートを**図11-2**に示します．この動作は単なるステート・マシンで実現できます．

● 高速インターフェースではI/Oバッファを使う

高速な入出力を行う場合にはI/Oバッファが不可欠です．今回のターゲットFPGAであるStratix IIのI/Oエレメント(IOE)には，OEレジスタ，OUTPUTレジスタ，INPUTレジスタがあります．これらを使用するような記述を行い，また設計ツールでオプションを指定します．

今回はZBT-SSRAM制御ブロックSSRAMから，双方向バッファIO_Buffers.v，出力バッファOUT_Buffers.v，入力バッファIN_Buffers.vを経由して外部入出力を行っています．また，米国Altera社のQuartus IIでは，Assignments Editerで，以下のオプションを有効にします．

- FAST_OUTPUT_REGISTER(出力または双方向のとき)
- FAST_INPUT_REGISTER(入力または双方向のとき)
- FAST_OUTPUT_ENABLE_REGISTER(双方向のとき)

I/Oレジスタを使わない記述では，SSRAMを166MHzで動作させることはできませんでした．

リード/ライト動作の検証で使用した疑似乱数発生回路などを含んだプロジェクトは，本誌のホームページからダウンロードできます．

よしだ・まさかず （有）プライムシステムズ

12 PCカード・インターフェース回路

川岡圭一

外部回路

- 回路図：図12-1
- 主要部品：CPLD

HDL記述

- Verilog HDL記述：リスト12-1
- 外部入力：pc104_a，pc104_d，pc104_smemw_n，pc104_smemr_n，pc104_iow_n，pc104_ior_n，pc104_reset_drv，pc104_sbhe_n，pc_ireq_n，pc_wait_n，pc_cd_n，bus_en_req，pc104_sysclk，pc104_d_out
- 外部双方向：pc_iord_n，pc_iowr_n，pc_ce1_n，pc_ce2_n，pc_we_n，pc_reg_n，pc_reset，pc_oe_n，
- 外部出力：pc104_iochrdy_n，pc104_irq3_n，pc104_irq4_n，pc104_irq5_n，pc104_irq6_n，pc104_irq7_n，pc_a，card_on_off，bus_en_n，databuff_245_oe_n，databuff_245_dir
- 内部入力：なし
- 内部出力：なし
- パラメータ：なし

評価環境

ISE WebPACK 7.1i，XC9572-TQ100

PCカードには，メモリ・カード，ATAカード，各種I/Oカードがあり，携帯機器などでよく使われています．また，メモリ・カードは，組み込み機器のストレージ・デバイスとして使われることもあります．

ここではPC/104バスに接続できるPCカード・インターフェース回路について説明します．

PCカード・インターフェース回路を**図12-1**に示します．

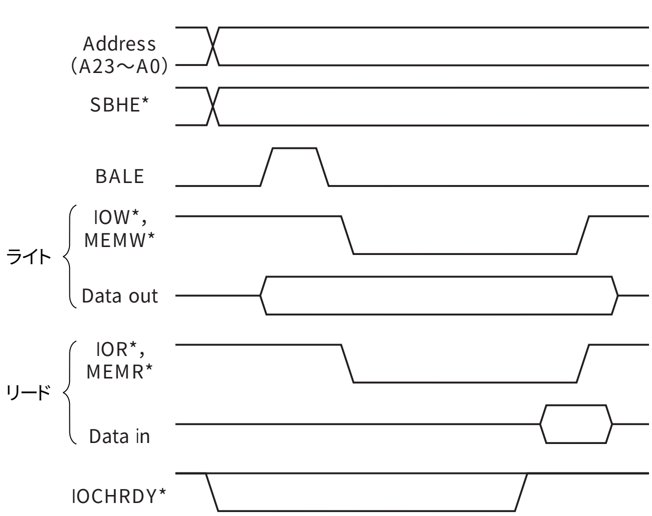


図12-2　PC/104インターフェースのリード/ライトのタイミング
PC/104インターフェースは，コネクタなどの仕様以外はISAと同じである．

米国Xilinx社のCPLD「XC9572-TQ100」にPC/104インターフェースとPCカード・コントローラ（米国Intel社のi83365簡易版）で，PC/104とPCカードのアドレス，および制御信号の変換を行っています．

また，PCカードの電源制御を行います．PCカードが挿入されていないときに，PCカードの電源をOFFに，出力信号をハイ・インピーダンスにします．これにより，挿抜時のPCカードの破壊を防止しています．

PC/104インターフェースは，コネクタなどの仕様以外はISAと同じです．基本的なリード/ライトのタイミングを**図12-2**に示します．

データ・ラッチのクロックにPC/104のIOWを使用します．クロックを必要とするのは，このデータ・ラッチ部のため，同期設計を意識していません．

割り込み線は，5本（IRQ＊3～7）のみ接続し，そのうちレジスタ設定で1本のみ選択します．割り込み要素は，PCカードからのIRQ＊のみです．

PCカード・コントローラは，Intel社のi82365を参考に，以下のような最低限の機能のみを実装しています．

- I/O WINDOWは一つのみ
- I/O空間は0x000～0x3FF
- MEMORY WINDOWは，アトリビューション・メモリ，

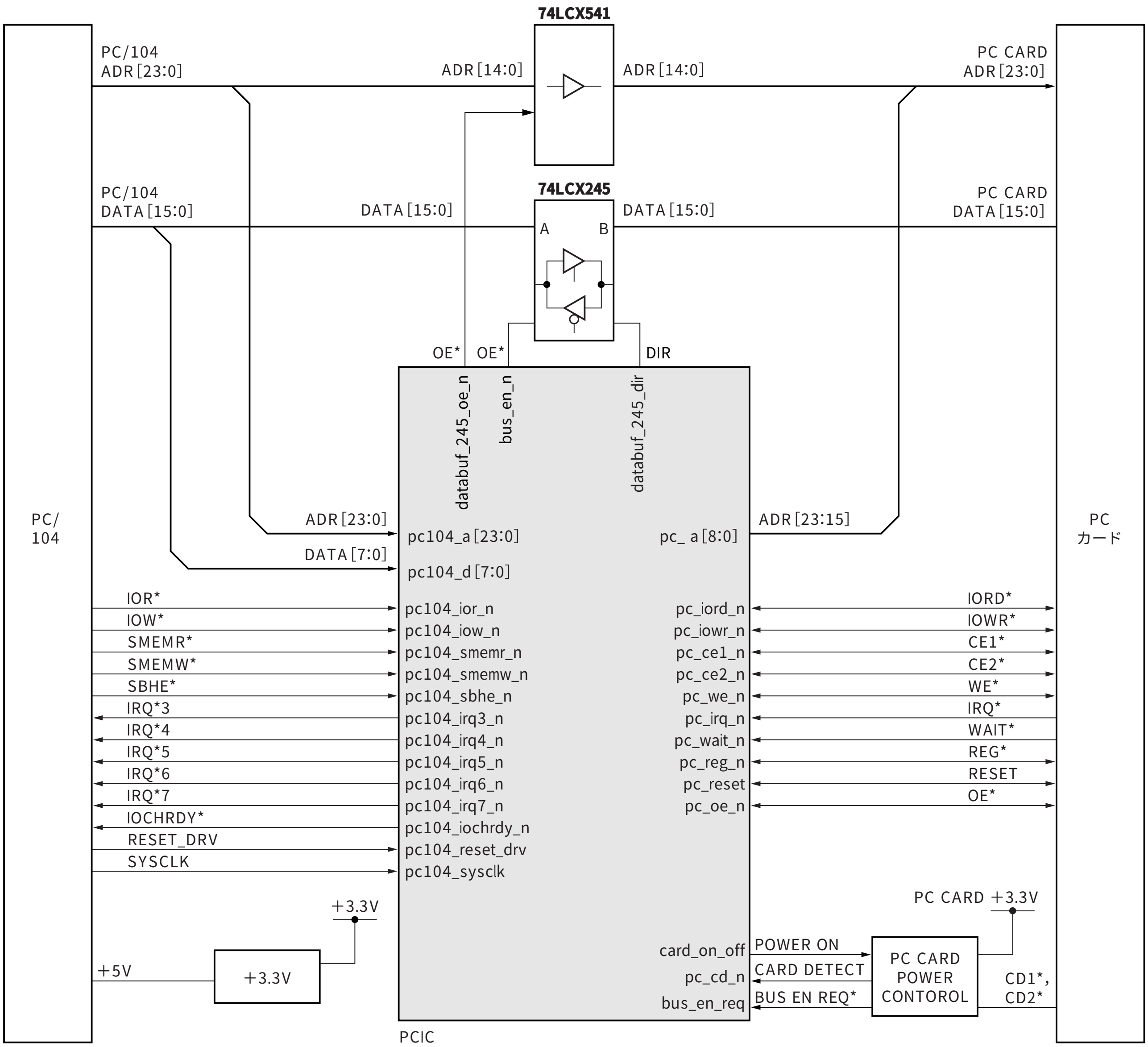


図12-1　PCカード・インターフェースの概略回路
中央のPCICが設計したPCカード・インターフェース・ブロックである．

コモン・メモリとも0x00C8000h～0x00C_FFFFh固定

- メモリ空間は0x00_0000～0xFF_FFFF
- リニアにアクセスできる範囲は16Kバイト．16Mバイト以上をアクセスするにはバンク切り替えを行う
- 割り込み要素は，PCカードからのIRQ*のみ
- PCカード電源のON/OFF
- カード実装状態の確認
- メモリ空間のアクセス方法（アトリビューション・メモリ，コモン・メモリ）

PCカード・インターフェース・ブロックPCICのVerilog HDL記述を**リスト12-1**に示します．

かわおか・けいいち
(株)アットマークテクノ

2

リスト12-1　PCカード・インターフェース・ブロックPCICのVerilog HDL記述

```
module PCIC (
      pc104_a ,pc104_d ,pc104_d_out ,pc104_smemw_n ,pc104_smemr_n ,pc104_iow_n ,
      pc104_ior_n ,pc104_reset_drv ,pc104_sysclk ,pc104_sbhe_n ,pc104_iochrdy_n ,
      pc104_irq3_n ,pc104_irq4_n ,pc104_irq5_n ,pc104_irq6_n ,pc104_irq7_n ,
      pc_wait_n ,pc_ireq_n ,pc_cd_n ,bus_en_req ,bus_en_n ,pc_ce1_n ,
      pc_ce2_n ,pc_we_n ,pc_oe_n ,pc_iord_n ,pc_iowr_n ,pc_reg_n ,
      pc_reset ,pc_a ,databuff_245_dir ,databuff_245_oe_n ,card_on_off
) ;

input [23:0]    pc104_a ;
input [7:0]     pc104_d ;
inout           pc104_d_out ;
input           pc104_smemw_n ;
input           pc104_smemr_n ;
input           pc104_iow_n ;
input           pc104_ior_n ;
input           pc104_reset_drv ;
input           pc104_sysclk ;
input           pc104_sbhe_n ;
output          pc104_iochrdy_n ;
output          pc104_irq3_n ;
output          pc104_irq4_n ;
output          pc104_irq5_n ;
output          pc104_irq6_n ;
output          pc104_irq7_n ;
input           pc_wait_n ;
input           pc_ireq_n ;
input           pc_cd_n ;
input           bus_en_req ;
output          bus_en_n ;
inout           pc_ce1_n ;
inout           pc_ce2_n ;
inout           pc_we_n ;
inout           pc_oe_n ;
inout           pc_iord_n ;
inout           pc_iowr_n ;
inout           pc_reg_n ;
inout           pc_reset ;
output[8:0]     pc_a ;
output          databuff_245_dir ;
output          databuff_245_oe_n ;
output          card_on_off ;

reg   [4:0]     index_register ;
reg             card_on_off ;
reg             pc_card_reset ;
reg   [2:0]     irq_number ;
reg             io_windows ;
reg   [4:0]     window0start_lo ;
reg   [1:0]     window0start_hi ;
reg   [4:0]     window0stop_lo ;
reg   [1:0]     window0stop_hi ;
reg             reg_active ;
reg   [8:0]     card_offset ;

//***************************************************
//************ ADDRESS 3E0 3E1 DECOODER ************
//***************************************************
wire index_register_decood = ( pc104_a[9:0] == 10'b11_1110_0000 ) ; // ADDRESS 3E0
wire data_register_decood = ( pc104_a[9:0] == 10'b11_1110_0001 ) ;  // ADDRESS 3E1


//***************************************************
//************ ADDRESS 3E0 3E1 DATA LATCH **********
//***************************************************
//wire reg_clr = pc104_reset_drv | pc_cd_n ;
wire reg_clr = pc104_reset_drv ;
always @(posedge pc104_iow_n or posedge reg_clr ) begin
if ( reg_clr ) begin
      index_register[4:0] <= 0 ;
      card_on_off <= 0 ;
      pc_card_reset <= 1 ;
      irq_number[2:0] <= 0 ;
      io_windows <= 0 ;
      window0start_lo[4:0] <= 0 ;
      window0start_hi[1:0] <= 0 ;
      window0stop_lo[4:0] <= 0 ;
      window0stop_hi[1:0] <= 0 ;
      reg_active <= 1 ;
      card_offset[8:0] <= 0 ;
end
```



写真12-1　制作したPCカード・インターフェース

```
else begin
//INDEX REGISTER
     index_register[4:0] <= ( index_register_decood ) ? pc104_d[4:0]  : index_register ;
//POWER AND RESETDRV
     card_on_off <= ( data_register_decood &  ( index_register == 5'h02 )) ? pc104_d[4] : card_on_off ; //030724
//INTERRUPT AND GENERAL CONTROL
     pc_card_reset <= ( data_register_decood &  ( index_register == 5'h03 )) ? pc104_d[6] : pc_card_reset ;
     irq_number[2:0] <= ( data_register_decood &  ( index_register == 5'h03 )) ? pc104_d[2:0] : irq_number ;
//ADDRES WINDOW ENABLE
     io_windows <= ( data_register_decood &  ( index_register == 5'h06 )) ? pc104_d[6] : io_windows ;
//IO ADDRESSMAPPING START STOP
     window0start_lo[4:0] <= ( data_register_decood &  ( index_register == 5'h08 )) ? pc104_d[7:3] : window0start_lo ;
     window0start_hi[1:0] <= ( data_register_decood &  ( index_register == 5'h09 )) ? pc104_d[1:0] : window0start_hi ;
     window0stop_lo[4:0] <= ( data_register_decood &  ( index_register == 5'h0a )) ? pc104_d[7:3] : window0stop_lo ;
     window0stop_hi[1:0] <= ( data_register_decood &  ( index_register == 5'h0b )) ? pc104_d[1:0] : window0stop_hi ;
     reg_active <= ( data_register_decood &  ( index_register == 5'h15 )) ? pc104_d[6] : reg_active ;
//PC CARD MEM ADDRESS = { WINDOW ADDRESS 9BIT(019h=0C8h) , ISA ADDRESS 15 BIT } + CARD MEMORY OFFSET ADDRESS[25:15] )
//CARD MEMORY OFFSET ADDRESS[19:15]
     card_offset[4:0] <= ( data_register_decood &  ( index_register == 5'h14 )) ? pc104_d[7:3] : card_offset[4:0] ;
//CARD MEMORY OFFSET ADDRESS[25:20]
     card_offset[8:5] <= ( data_register_decood &  ( index_register == 5'h15 )) ? pc104_d[3:0] : card_offset[8:5] ;
end
end


//***************************************************
//************ OUTPUT DATA ENABLE ******************
//***************************************************
wire pc104_oe = data_register_decood & ~pc104_ior_n ;
assign pc104_d_out = ( pc104_oe )? ~pc_cd_n : 1'bz ;


//***************************************************
//************ PC104 => PC CARD  ******************
//***************************************************

// MEMRY EREA ADDRESS WINDOW 0x0C8
wire mem_window_comp = ( pc104_a[23:15] == 9'b000011010 ) ;  // ISA 0x0C8h ( 23:15 = 019h)

wire io_window_comp_sta_hi = ( window0start_hi == pc104_a[9:8] ) ;
wire io_window_comp_sta_lo = ( window0start_lo <= pc104_a[7:3] ) ;
wire io_window_comp_stp_hi = ( window0stop_hi == pc104_a[9:8] ) ;
wire io_window_comp_stp_lo = ( window0stop_lo >= pc104_a[7:3] ) ;

wire io_window_comp = io_window_comp_sta_hi & io_window_comp_sta_lo & io_window_comp_stp_hi & io_window_comp_stp_lo ;

wire pc_ce1_w_n = ~((( ~pc104_smemw_n | ~pc104_smemr_n ) & mem_window_comp )
                   | (( ~pc104_iow_n | ~pc104_ior_n ) & io_window_comp & io_windows )) ;

assign pc_ce1_n = ( ~bus_en_req )? pc_ce1_w_n : 1'bz ;
assign pc_ce2_n = ( ~bus_en_req )? ~( ~pc_ce1_w_n  & ~pc104_sbhe_n ) : 1'bz ;
assign pc_we_n = ( ~bus_en_req )? ~( ~pc_ce1_w_n  & ~pc104_smemw_n ) : 1'bz ;
assign pc_oe_n = ( ~bus_en_req )? ~( ~pc_ce1_w_n  & ~pc104_smemr_n ) : 1'bz ;
assign pc_iord_n = ( ~bus_en_req )? ~( ~pc_ce1_w_n  & ~pc104_ior_n ) : 1'bz ;
assign pc_iowr_n = ( ~bus_en_req )? ~( ~pc_ce1_w_n  & ~pc104_iow_n ) : 1'bz ;
assign pc_reg_n = ( ~bus_en_req )? ~(( ~pc_ce1_w_n  & reg_active ) | ~pc_iowr_n | ~pc_iowr_n ) : 1'bz ;
assign pc_reset = ( ~bus_en_req )? ~pc_card_reset | pc104_reset_drv : 1'bz ;


assign pc_a[8:0] = ( bus_en_req )? 9'bzzzzzzzzz : ( ~pc_iord_n | ~pc_iowr_n )? 0 : card_offset[8:0] ;


//***************************************************
//************ PC CARD => PC104  ******************
//***************************************************
assign pc104_iochrdy_n = ( ~pc_wait_n & io_window_comp & ~bus_en_req )? 1'b0 : 1'bz ;
assign pc104_irq3_n = (( irq_number == 3 ) & ~bus_en_req & ~pc_ireq_n )? 1 : 0 ;
assign pc104_irq4_n = (( irq_number == 4 ) & ~bus_en_req & ~pc_ireq_n )? 1 : 0 ;
assign pc104_irq5_n = (( irq_number == 5 ) & ~bus_en_req & ~pc_ireq_n )? 1 : 0 ;
assign pc104_irq6_n = (( irq_number == 6 ) & ~bus_en_req & ~pc_ireq_n )? 1 : 0 ;
assign pc104_irq7_n = (( irq_number == 7 ) & ~bus_en_req & ~pc_ireq_n )? 1 : 0 ;

//***************************************************
//************ DATA BAS CONTROL(LVC245) ************
//***************************************************

assign databuff_245_dir = ( pc104_smemr_n & pc104_ior_n ) | bus_en_req ;
assign databuff_245_oe_n = ~( ~pc_ce1_w_n & ~bus_en_req ) ;
assign bus_en_n = bus_en_req ;

endmodule
```

13 I²C-busインターフェース回路

河野 崇

外部回路
- 回路図：図13-3
- 主要部品：FET，抵抗

HDL記述
- VHDL記述：本誌ホームページからダウンロード可能

i2cMaster
- 外部入力：SCL，SDA
- 外部出力：mstSCL，mstSDA
- 内部入力：MReq，Mi2cAdrs，MRnW，Mcont，MStb，MwrData，MwrTerm，MwrDRdy，MrdTerm，MrdDRdy，Spre，Spos，Tlow，Thig，Dhol，nRST，syncRST，CKE，CLK
- 内部出力：Mbusy，MwrDReq，MrdDReq，MrdData，srpBFree，srpBObsc，srpAdrs，srpWrite，srpArbt，srpAbort
- パラメータ：なし

i2cSlave
- 外部入力：SCL，SDA
- 外部出力：slvSCL，slvSDA
- 内部入力：SAck，SwrTerm，SwrDRdy，SrdData，SrdDRdy，Dsup，nRST，syncRST，CKE，CLK
- 内部出力：Sreq，Si2cAdrs，SRnW，SwrDReq，SwrData，SrdDReq，srpTerm，srpAbort，srpBFree，srpBObsc
- パラメータ：なし

評価環境

Quartus II Web Edition 5.1，Stratix

I²C-bus（Inter IC bus）はオランダPhilips Semiconductors社によって策定されたシステム内伝送用のシリアル・バスです．SDA（Serial Data Line）とSCL（Serial Clock Line）の2本の信号線のみを用いたシンプルな構成でありながら，複数のマスタ・デバイス（データ転送を開始するデバイス）の接続が可能です（**図13-1**）．転送速度は，標準モード（Standard-mode；100kbps），ファースト・モード（Fast-mode；400kbps），高速モード（High-speed mode；4.3 Mbps）の3種類があります．従来，I²C-busの使用にあたってはライセンス契約とロイヤリティの支払いが必要でしたが，2004年8月にPhilips社の特許が失効し，不要になりました．

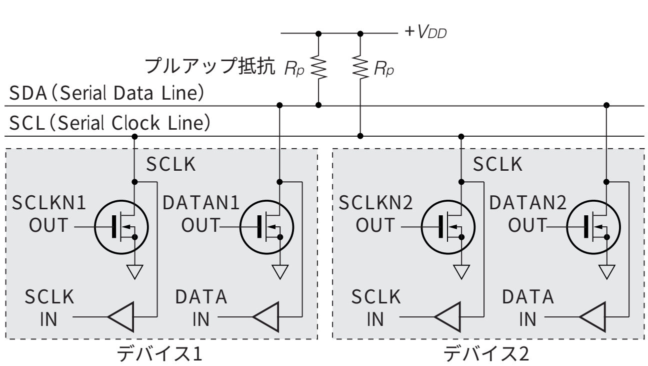


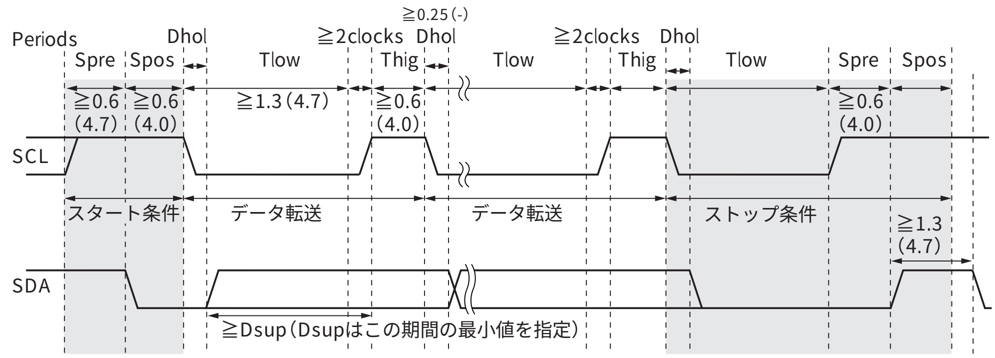
図13-1　I²C-busの接続
標準モードとファースト・モードの接続例を示す．SDA（Serial Data Line）とSCL（Serial Clock Line）の2本の信号線のみを用いたシンプルな構成である．

● 2本の信号線だけで通信する

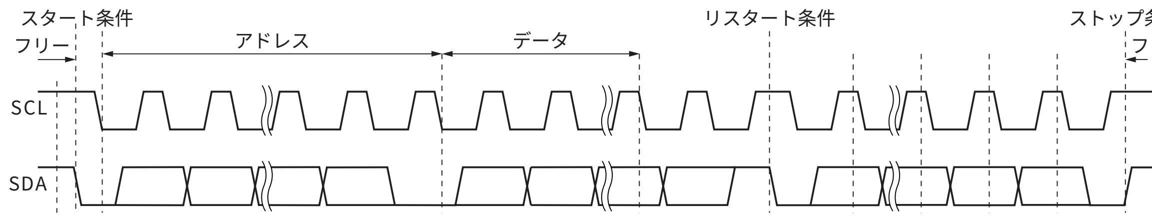
図13-2（a）にI²C-busの動作タイミングを示します．SCLが“H”の間にSDAが立ち下がる動作で，トランザクションの開始を宣言します（スタート条件）．データの転送は，SCLが“H”の間にSDAを“L”または“H”で固定することによって行います（データ転送）．SCLが“H”の間にSDAが立ち上がる動作で，トランザクションの完了を宣言し，バスをフリーな状態（SCL，SDAともに“H”）に解放します（ストップ条件）．

トランザクションを開始したいマスタ・デバイスは，バスがフリーであることを確認してからスタート条件によりバス使用権を確保します．そして，通信したいデバイスのアドレス（7ビット）と方向（“H”＝リード/“L”＝ライト）を送信し，応答（ACK＝“L”）があった場合には8ビットごとにデータの転送を行います．データの転送が完了したら，ストップ条件によりバスの使用権を解放するか，続けてスタート条件により新しいトランザクションを開始します（**図13-2（b）**）．

図13-2
I²C-busのタイミング図
マスタ・デバイスは，バスがフリーな状態であることを確認し，スタート条件でトランザクションの開始を宣言する．そしてデータの転送を行う．ストップ条件でトランザクションの完了を宣言し，バスをフリーな状態に解放する．図中の数字は，仕様で示されているファースト・モードと標準モード（かっこ内）の時間で，単位はμs．



（**a**）基本タイミングとパラメータ



（**b**）データ転送手続き

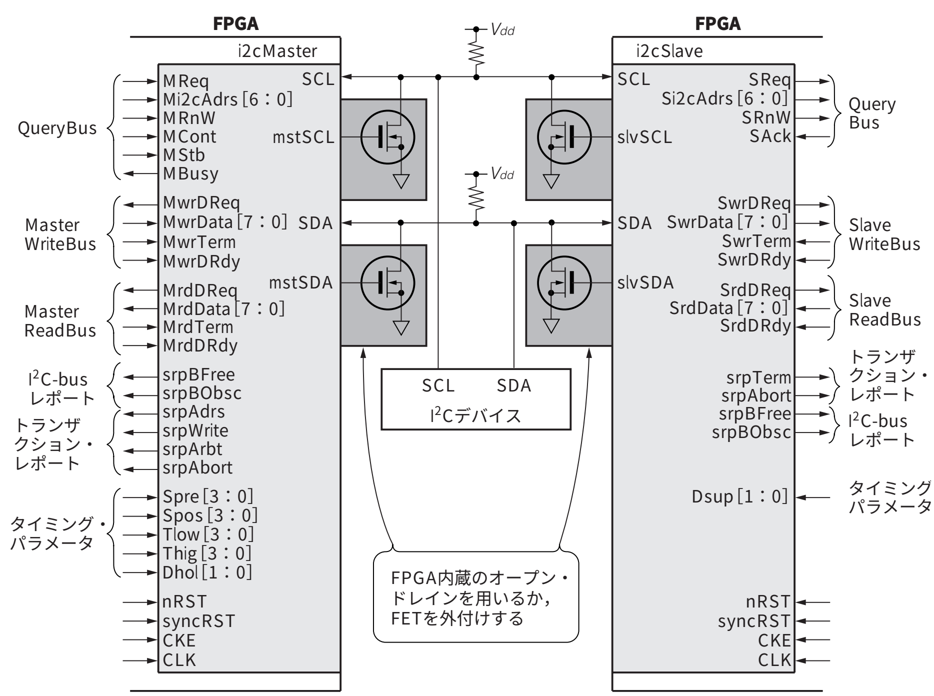
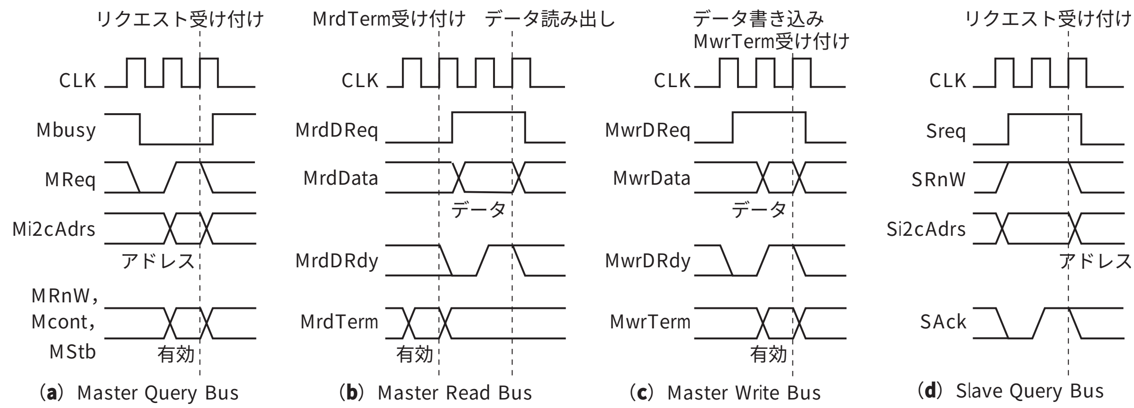


図13-3
マスタ機能を実現するi2cMasterとスレーブ機能を実現するi2cSlaveを用いたI²C-busの回路
プルアップ抵抗の値はV_{dd}の値や配線長などに応じて決定する．V_{dd}＝3.3Vで1kΩ程度が目安となる．なお，i2cMasterとi2cSlaveは，高速モードと10ビット・アドレッシングには対応していない．

マスタ機能を実現するi2cMasterとスレーブ（マスタの開始したデータ転送に応答する）機能を実現するi2cSlaveを用いたI²C-busの回路を**図13-3**に示します．SCL，SDAはI²C-busからの入力，mstSCLとslvSCL，mstSDA，slvSDAはI²C-busへの出力です．FPGAの多くはオープン・ドレイン・ドライバを内蔵しているのでそれを用います．

CKEはSCL，SDAをドライブ/サンプルするためのクロック・イネーブル信号です．**図13-2(a)**に示すとおり，パラメータ入力（i2cMasterのSpre，Spos，Tlow，Thig，Dholとi2cSlaveのDsup）はI²C-busドライブ時のタイミング・パラメータで，それぞれCKEの回数－1を指定します．まず，CKEの周期はSCL，SDAの遷移時間の最大値（ファ



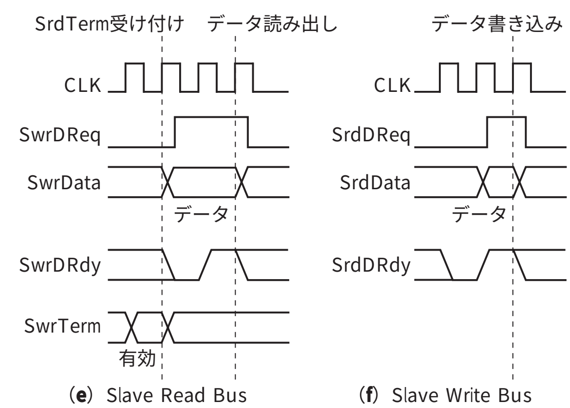


図13-4 i2cMasterとi2cSlaveの動作

i2cMasterとi2cSlaveのFPGA内部ポートの動作を示す．

ースト・モードでは300ns，標準モードでは1000ns）を3で除した値以上にします．ファースト・モードでは100ns，標準モードでは400ns前後がよいでしょう．例えばファースト・モードではSpre＝6，Spos＝6，Tlow＝13，Thig＝4，Dhol＝1，Dsup＝2と指定します．

● マスタ・コントローラi2cMasterの動作

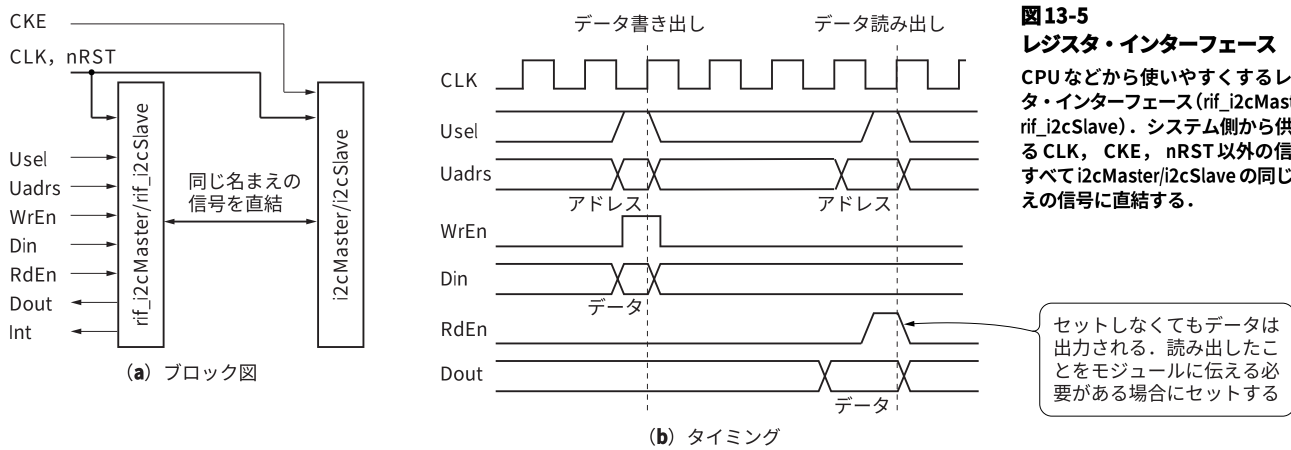
i2cMasterのQuery Busはトランザクション開始を指示するためのものです．Mi2cAdrs，MRnW，Mcont，MStbを設定したうえでMReqを‘1’にすると，MBusyが‘0’のクロックで受け付けられます（**図13-4（a）**）．MRnWが‘1’でリード転送，‘0’でライト転送です．MContが‘1’のときはストップ条件ではなくリスタート条件であり，次のトランザクションへつなぎます（バスを解放してやる）．MStbが‘1’のときはStart Byteを発行します．

Master Read Busはリード・トランザクションのデータを受け取るためのものです．I^2C-busからデータが読み込まれるとMrdDataに提示され，同時にMrdDReq信号が‘1’になります．MrdDRdyを‘1’にすることでデータ受け取り完了を知らせます（**図13-4（b）**）．この応答が遅れるとi2cMasterは次のデータの読み出しを開始できず，I^2C-busを占有したまま待ちます．長期間MrdDRdyが遅れる場合はあらかじめMrdTermを‘1’にしてトランザクションをいったん終了させます．MrdTermはMrdDReqが‘1’になる（データが到着する）前にセットしなければなりません．

Master Write Busはライト・トランザクションのデータを送出するためのものです．I^2C-busに送出するデータが必要になったときにMwrDReqが‘1’になります．書き込みデータをMwrDataにセットすると同時にMwrDRdyを‘1’としてやることでデータを与えます（**図13-4（c）**）．MwrDReqとMwrDRdyの両方が‘1’のクロックでデータが読み込まれるため，あらかじめMwrDRdyを‘1’にセットしておくことが可能です．MwrDRdyが遅れるとi2cMasterはI^2C-busを占有したまま待ちます．トランザクションの完了は，最終データを与えるのと同じタイミングでMwrTermを‘1’にセットしておくことで行います．

srpAdrs，srpWrite，srpArbt，srpAbortは転送の終了状態を表す信号です．それぞれ，「アドレスに応答がなかった」，「書き込み中にスレーブ・デバイスの要求でトランザクションが中断された」，「アービトレーションに負けてバスの使用権を失った」，「トランザクション中に異常な競合が発生した」，という意味です．これらの信号はいずれも，トランザクションが終了してMbusyが‘0’になる数クロック前に有効となり，次のトランザクション要求が受け付けられる（MReqが‘1’でMbusyが‘0’）と同時にリセットされます．

srpBFreeとsrpBObscは現在のバスの状態を表しています．前者はI^2C-busが解放されていることを示し，後者はリセット直後などでバスの状態がわからないことを示しています．



(a) ブロック図
(b) タイミング

図13-5
レジスタ・インターフェース
CPUなどから使いやすくするレジスタ・インターフェース（rif_i2cMaster，rif_i2cSlave）．システム側から供給するCLK，CKE，nRST以外の信号はすべてi2cMaster/i2cSlaveの同じ名まえの信号に直結する．

● スレーブ・コントローラi2cSlaveの動作

i2cSlaveのQuery Busはアドレス・デコーダを接続するためのものです．スタート条件を検出したi2cSlaveはアドレスと転送方向を読み込み，Si2cAdrsとSRnWに提示すると同時にSreqを'1'にします（**図13-4(d)**）．応答する場合は，一定時間内（最小値は2クロックだが多くの場合はもっと長い）にSAckを'1'にセットします．SreqとSAckがともに'1'のクロックで応答が受け付けられます（次のSreqが来るまでにSAckをリセットしておく）．

Slave Write Busはライト・トランザクションによりスレーブ・デバイスに送られてくるデータを受け取るためのものです．i2cSlaveはデータを受け取るとSwrDataをセットし，同時にSwrDReqを'1'にします．SwrDRdyを'1'にすることでデータ受け取りを知らせます（**図13-4(e)**）．この応答が遅れると，i2cSlaveはI^2C-bus仕様で規定されているシンクロナイゼーションという方法でウェイトをかけます．これ以上データを受け取れない場合は，SwrTermをセットすることによってトランザクションの中断を要求することができます．この信号もMrdTermと同様のタイミングでセットしておく必要がありますが，FIFO（first-in first-out）を直に接続する場合のことを考え，SwrDRdyが'1'になる以前，かつSReqが'1'になってから数クロック（最小2クロック）以内であれば受け付けられるようにしてあります．

Slave Read Busはリード・トランザクションによってスレーブ・デバイスが送らなければならないデータを書き込むためのものです．I^2C-busバスに送出するデータが必要になったときにSrdDReqが'1'になります．書き込みデータをSrdDataにセットすると同時にSrdDRdyを'1'としてやることで，データを与えます（**図13-4(f)**）．SrdDReqとSrdDRdyの両方が'1'のクロックでデータが読み込まれるので，あらかじめSrdDRdyを'1'にセットしておくことが可能です．SrdDRdyが遅れるとi2cSlaveはシンクロナイゼーションによってウェイトをかけます．リード・トランザクションではスレーブ・デバイスが転送中断を要求する手段はありません．

srpTermとsrpAbortは転送の終了状態を表す信号です．それぞれ，「正常終了」，「異常なスタート/ストップ条件を検出することにより異常終了」，を意味します．これらの信号はSAckが'1'になるとリセットされます．

2

● CPUから使いやすくするレジスタ・インターフェース

レジスタ・インターフェース（rif_i2cMaster，rif_i2cSlave）があると，CPUなどからも使いやすくなります．簡単なブロック図を**図13-5(a)**に信号のタイミング図を**図13-5(b)**に示します．

今回設計したI^2C-bus制御ブロックは，温度センサの「LM92（米国National Semiconductor社）」を接続して動作を確認しています．しかし，すべての動作を完全に検証するまでには至っていません．今後のアップデートは筆者のホームページ（http://www.digicat.info/dwm/）で行う予定です．

参考・引用*文献

(1) Philips Semiconductors I^2C-bus Information, http://www.semiconductors.philips.com/markets/mms/protocols/i2c/.
(2) I^2C Bus Technical Overview and FAQ, http://www.esacademy.com/faq/i2c/index.htm.
(3) I^2Cバス，http://www.necel.com/ja/faq/mi78k/_78iic.html.

こうの・たかし
独立行政法人 科学技術振興機構